UMLによる組込みソフトウェア設計の検証支援環境の開発





横川　智教[†]　天嵜　聡介[†]　佐藤　洋一郎[†]　有本　和民[†]　宮崎　仁[‡]

情報システムは年々複雑化しており，その信頼性を保証することが大きな課題となっている．上流工程で網羅的に設計を検証できるモデル検査の優位性が認識されているが，専門知識やノウハウの習得の難しさが問題となっている．本論文ではモデル検査による設計検証を支援するツールを提案する．本ツールでは開発者が普段記述している設計文書から自動でモデル検査に必要な文書を生成し設計を検証できる．協力企業で実際作成された設計文書へ本ツールを適用した結果，その有用性が確認できた．

Tool Support for Verification Environment of Embedded Software Design with UML

Tomoyuki Yokogawa [†], Sousuke Amasaki [†], Yoichiro Sato [†], Kazutami Arimoto [†] and Hisashi Miyazaki [‡]

Information systems get more complicated recently, and assuring their reliability has been an urgent problem. Model checking is known as one of advantageous technology because of its nature: exhaustive and automatic verification available in early development phase. However, it is also known for difficulty in learning special knowledge. This paper proposed a tool supporting model checking on software design artifacts. This tool can perform automatic verification on software design artifacts written in a well-known notation by translating them into specialized format files suitable for a model checking software and performing verification on the files. In cooperation with a software development company, we confirmed that this tool can help software reliability assurance.

1. はじめに

情報システムは年々複雑化しており，どのように信頼性を保証するかが大きな課題となっている．クリティカルな分野のソフトウェアでは1件の不具合が社会に及ぼす影響が甚大であるため，特に品質保証が重要視される．ソフトウェア開発における修正のコストは開発工程の後半に進むほど大きくなるため，設計検証に基づいた上流工程での品質保証が求められる．

設計検証による品質保証のために有効な手段と考えられるのがモデル検査 [Clarke1999] である．モデル検査では，ソフトウェアの振る舞いを有向グラフによってモデル化し，グラフの網羅的探索により求める特性が満たされるか否かを自動的に検証することが可能である．モ

【脚注】
† 岡山県立大学 情報工学部
‡ 川崎医療福祉大学 医療技術学部

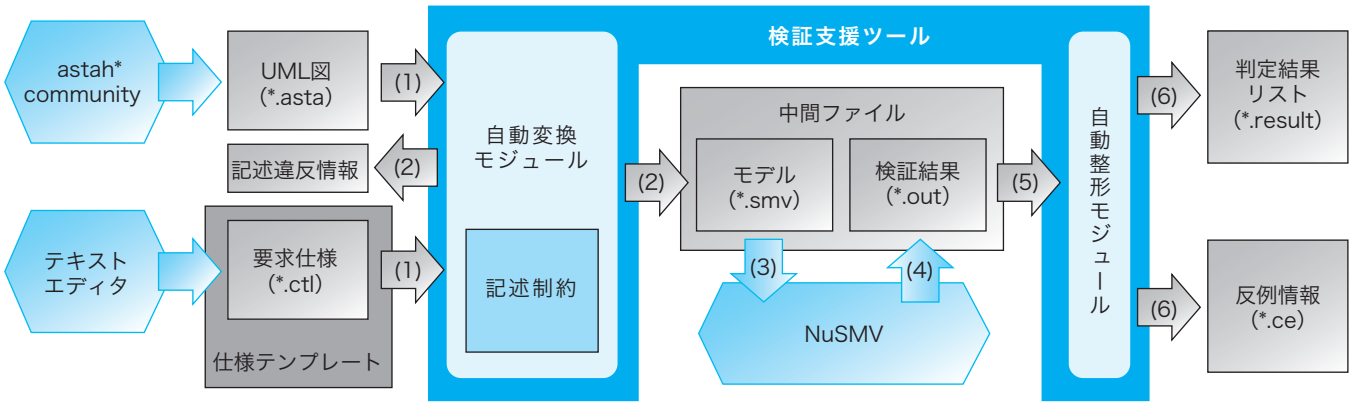

図 1　本ツールを用いた設計検証の枠組み

デル検査を用いることで設計の矛盾や仕様との不整合を自動でかつもれなく検出することが可能となる．

モデル検査を実施するためのツールは SMV[McMillan 1993] や SPIN[Holzmann1997] をはじめとして数多くのものが公開されている．しかしながら，モデル検査を設計検証に導入するためには，ソフトウェアの設計文書をツール固有のモデル化言語で記述しなければならない．モデル化言語の記法は，ソフトウェア開発者が通常使用している設計文書とは大きな隔たりがあり，設計文書の記述規則は実務的な利便性に重きが置かれる一方で，モデル化言語ではシステムの振る舞いを厳密に記述するための意味論が定められている．したがって，ソフトウェアの設計文書をもとにモデル化言語による記述（以下，モデルという）の作成を行うには専門的な知識やノウハウが必要となる．

そこで本論文では，モデル検査技術を組込みソフトウェアの設計検証に導入する上でのコスト削減という課題の解決を目的として，設計文書からモデルを自動生成する検証支援ツールを開発する．本ツールでは，開発現場において広く用いられている設計記法の一つである UML(Unified Modeling Language)[UML] を対象として，モデルの自動生成を行う．また，モデル検査ツールとして，記号モデル検査アルゴリズム [Burch1990] に基づく高速な検証が可能である NuSMV[NuSMV] を用いる．本研究の主な貢献点は以下の 3 つである．

(1)　UML 描画ツール astah* community[astah] との連携
(2)　NuSMV の入力モデルへの自動変換
(3)　ツールの公開 [Tool]

本ツールにより，UML 描画ツールで作成された設計文書をモデル検査ツールの入力モデルへと自動変換を行うことが可能となり，モデル検査を設計検証へと導入する上でのコストが大幅に削減できる．これにより，上記の課題が解決される．

本論文では，協力企業から提供を受けたソフトウェア設計文書に対して本ツールを適用し，検証を行っている．適用実験を通して，本ツールの導入により，新たなコストを要することなくモデル検査による設計検証が実現できることを確認している．さらに，NuSMV によって検出された反例が設計誤りの修正に有効であることを併せて示している．

2.　検証支援ツール

2.1. 概要

本ツールは，astah* シリーズのソフトウェアの 1 つで無償利用が可能である UML モデリングツールである astah* community(以下，単に astah* という) で記述された UML によるソフトウェア設計文書とソフトウェアが満たすべき仕様を入力とし，NuSMV に対応するモデルに変換し出力する．仕様の記述はあらかじめ定められたテンプレートを用いて行われる．また，本ツールは NuSMV による検証結果を整形し出力する機能を有する．

UML はソフトウェアの異なる側面を記述する 13 種類の図 (UML 図) から構成されるが，本ツールでは，組込みソフトウェアの振る舞いを記述するために利用され，利用頻度の高い状態マシン図を取り扱う．本ツールはコンソールアプリケーションとして Java を用いて実装されており，コマンドラインで起動する．

2.2. 設計検証の枠組み

本ツールを用いた設計検証の枠組みを図 1 に示す．設計検証は以下の手順で実施される．

ステップ 1.　ユーザは astah* で作成した状態マシン図 (*.asta) と，仕様テンプレートに基づいて記述した要求仕様 (*.ctl) を作成し，本ツールに入力する．この ctl ファイルはテキスト形式で作成する．

ステップ 2.　本ツールにより NuSMV の入力となるモデル (*.smv) が生成される．このとき，asta ファイル

として入力された状態マシン図が記述制約を満たさなかった場合はエラーとなり，記述違反箇所の情報が出力される．

ステップ 3. ユーザは smv ファイルを NuSMV へ入力して検証を行う．

ステップ 4. NuSMV によって状態マシン図が要求仕様を満たすか否かについて検証した結果 (*.out) が出力される．

ステップ 5. ユーザは out ファイルを本ツールに入力する．

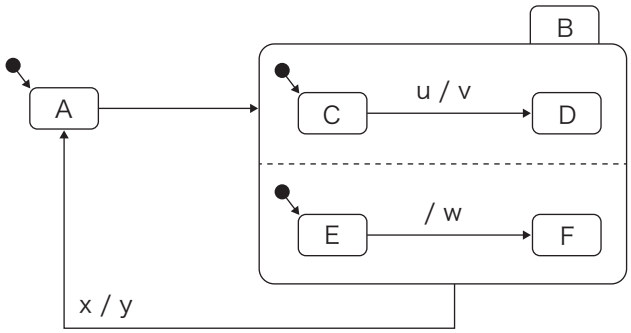

図 2　合成状態をもつ状態マシン図

表 1　状態マシン図に関する制約

	記法	可否
状態関連	単純状態	○
	合成状態	×
	直交状態	×
	擬似状態	
	初期擬似状態	○
	履歴	×
	ジョイン・フォーク	×
	接合点	○
	選択擬似状態	○
	入場点，退場点	×
	終了状態	○
	入退場アクション	×
	状態内アクション	×
遷移関連	変数	△
	アクション	
	メッセージ送信	○
	変数更新	△
	イベント	○
	ガード条件	
	活性状態に関する条件	○
	変数に関する条件	△

○： 記述できる
△： 一部記述できる
×： 記述できない

ステップ 6. 本ツールにより検証結果が整形され，要求仕様が満たされるか否かに関する判定結果リスト (*.result) が生成される．ここで，ステップ 4 において NuSMV による検証の結果，要求仕様が満たされなかった場合は，NuSMV が生成した反例に関する情報 (*.ce) を出力する．result ファイルおよび ce ファイルはテキスト形式で保存される．判定結果リストからは ctl ファイルとして入力された要求仕様のどれが満たされ，どれが満たされなかったについての情報が得られる．また，反例からは，要求仕様を満たさないような動作系列についての情報を得ることができる．

2.3. UML の記述制約

UML は実務的な利便性に重きが置かれており，記述上の意味論について解釈が分かれる部分が多い．例えば，合成状態をもつ状態マシン図では，遷移の交互実行の解釈は設計者の意図と必ずしも一致しない．例として，図 2 に示すような合成状態をもつ状態マシン図において，状態 A から合成状態 B へと遷移した際は状態 C と状態 E がアクティブとなる．その後イベント u を受信すると状態 C から D へ遷移するが，その際に並行してイベントをもたない状態 E から F への遷移が行われるか否かの解釈は設計者によって異なる．そこで，本ツールでは対象とする UML の記法に制約を与え，意味論が一意に定まる記述のみを自動変換の対象とする．

状態マシン図に対する記述制約を表 1 に示す．ここで可否が△となっているものは，以下のように部分的に制約が加えられていることを示す．まず，変数は整数型およびブール型に限るものとし，アクションにおける変数の更新については，右辺に変数および定数の四則演算をもつ代入文によってのみ行われる．そして，ガード条件上での変数に関する条件は，変数と整数値に関する線形制約に限るものとする．線形制約とは，「mx ～ n」の形 (x は変数，m,n は整数，～は＜ , ＞ , ＝のいずれか) をもつ式と，否定，論理和および論理積からなる論理式のことを指す．

表 2　仕様テンプレートと対応する CTL 式

No	仕様テンプレート	CTL 式
1	safe(変数 = 状態)	AG !(変数 = 状態)
2	safe(メッセージ)	AG !(メッセージ =TRUE)
3	safe(変数，値)	AG !(変数 = 値)
4	live(変数 = 状態)	AF (変数 = 状態)
5	live(メッセージ)	AF (メッセージ =TRUE)
6	live(変数，値)	AF (変数 = 値)
7	reachable(変数 = 状態)	EF (変数 = 状態)
8	reachable(メッセージ)	EF (メッセージ =TRUE)
9	reachable(変数，値)	EF (変数 = 値)

2.4. 仕様テンプレート

モデル検査を用いて設計検証を行うためには，対象となるシステムが満たすべき仕様を時相論理 [Pnueli1977] を用いて記述する必要がある．NuSMV では時相論理の一種である CTL(Computation Tree Logic)[Clarke1981] を用いる．ここで，時相論理とは「いつかある状態に到達する」や「常にある性質が満たされる」といったシステムの性質を，状態の遷移や時間の経過の観点から記述するための論理体系である．しかしながら，ソフトウェアが満たすべき仕様を，時相論理式として記述するためには，論理学や離散数学などの専門的な知識を要する．そこで，ソフトウェアが満たすべき仕様を入力するためのテンプレートを提供する．これにより，専門的な知識を有しないユーザであっても記述が容易となる．

本ツールで用いる仕様テンプレートを表 2 に示す．仕様テンプレートは，状態マシン図の重要な要素である状態，メッセージ，そして変数に関する特性を記述することが可能であり，特性として利用頻度が高い安全性 (safe)，活性 (live)，そして到達可能性 (reachable) の 3 つを記述することが可能である．

2.5. モデルへの変換

図 3 に NuSMV の入力言語によるモデルの基本構成を示す．モデルは変数宣言部，状態遷移系記述部および検査式記述部からなる．変数宣言部では検査対象となるモデルの要素を変数として宣言する．状態遷移系記述部では，モデルの各要素が，他の要素の値に基づいて定義される遷移条件に依存して，どのように変化するかを記述する．そして検査式記述部では，満たすべき性質を CTL による検査式で記述する．本ツールでは，変数はブール型と列挙型の二通りのみを用いる．以下に，本ツールにおける状態マシン図から状態遷移記述への変換および仕様テンプレートから検査式記述への変換について述べる．

状態マシン図から状態遷移記述への変換

状態遷移記述部では，状態マシン図の遷移の発火による状態，メッセージおよび変数の値の変化を case 文で記述する．状態の変化は以下のルール 1 ～ 3 に従って記述し，メッセージおよび変数の変化はルール 4 に従って記述する．

ルール 1. 実行可能な遷移が 1 つのみならば，その遷移先の状態へと変化する．

ルール 2. 実行可能な遷移が 2 つ以上存在するならば，いずれかの遷移先の状態へと変化する．

ルール 3. 実行可能な遷移が存在しなければ，状態は変化しない．

ルール 4. メッセージおよび変数の値を変化させる遷移のいずれかが実行されたときかつそのときのみ，それに従って値を変化させる．

仕様テンプレートから検査式記述への変換

検査式記述部では，テンプレートを用いて記述された仕様を CTL による検査式として記述する．仕様テンプレートから得られる CTL 式を表 2 に併せて示す．まず，安全性については CTL の AG 演算子によって記述する．AG 演算子は，全ての実行系列において常に成り立つような性質を表す．よって，「AG !(式)」により，「与えられた式が決して成り立たない」という安全性に関する特性を表すことができる．ここで「!」は論理否定を表す演算子である．次に，活性については AF 演算子によって記述する．AF 演算子は，全ての実行系列においていつか成り立つような性質を表す．よって，「AF (式)」により，「与えられた式がいつか必ず成り立つ」という活性に関する特性を表すことができる．最後に到達可能性に

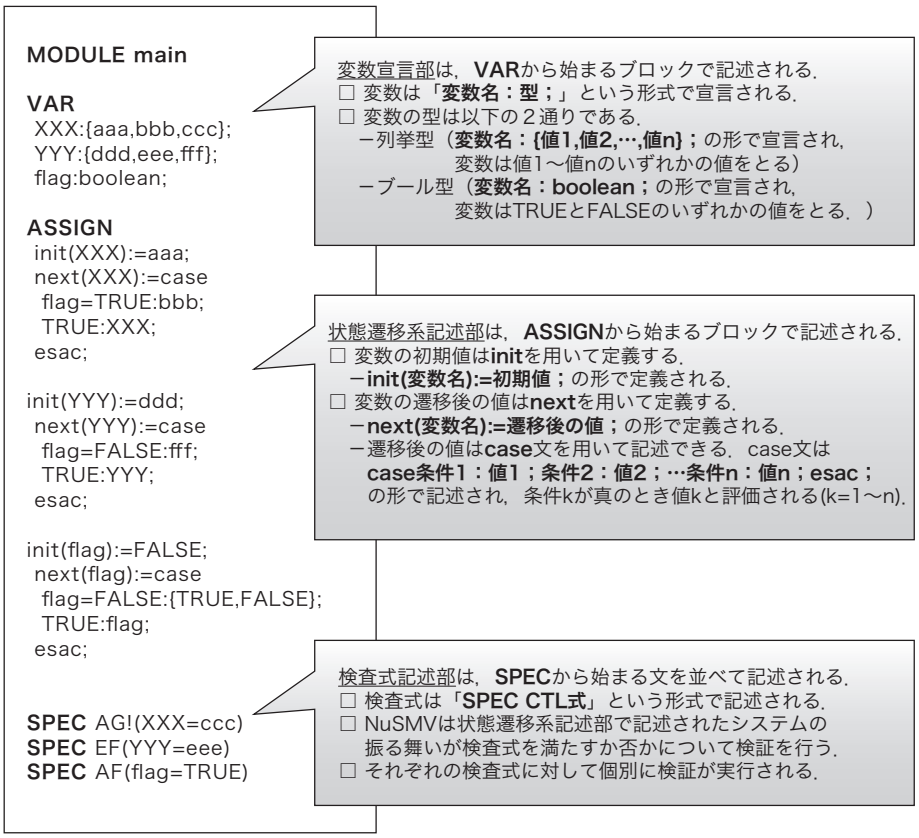

図３　モデルの基本構成

ついては EF 演算子によって記述する．EF 演算子は，いずれかの実行系列においていつか成り立つような性質を表す．よって，「EF (式)」により，「与えられた式がいつか成り立つ可能性がある」という到達可能性に関する特性を表すことができる．

3. 適用事例

3.1. 概要

協力企業から提供された状態マシン図に対して本ツールを適用し NuSMV による検証を行った．この状態マシン図は，実際の開発で用いられた状態遷移表などをもとに作成されたものである．適用対象のシステムは，店舗従業員向けの商品供給指示システム（R-CDS）である．R-CDS の概要は以下のとおりである．

- 売場従業員 (以下，従業員) と商品管理者 (以下，管理者) との通話システムである．
- 管理者は従業員に什器への商品の補充指示を発令し，従業員は補充の結果や現場の状況報告等を行う．従

表３　R-CDS の状態遷移表

イベント	状態			
	wait	send	receive	com
report	send			
accept			com MD_A=MD_A+1	
stop_report		wait		
reject			MD_A=0: wait MD_A=1: com	
finish_A				MD_A=1 : wait MD_A=MD_A-1
order				MD_A=1 : receive
response		com MD_A=1		
stop_order			MD_A=0 : wait MD_A=1 : com	
non_ response		wait		
finish_Ctl				MD_A=1 : wait MD_A=MD_A-1

表４　仕様テンプレートで記述した検査特性

検査項目	テンプレートで記述した仕様
状態の到達可能性	reachable(Terminal_A = send)
	reachable(Terminal_A = receive)
	reachable(Terminal_A = com)
	reachable(Terminal_B = send)
	reachable(Terminal_B = receive)
	reachable(Terminal_B = com)
通信量の到達可能性	reachable(MD_A, 1)
	reachable(MD_A, 2)
	reachable(MD_B, 1)
	reachable(MD_B, 2)
通信量の安全性	safe(MD_A, 3)
	safe(MD_A, -1)
	safe(MD_B, 3)
	safe(MD_B, -1)
管理者側端末の表示の到達可能性	reachable(Ctl_Monitor = surplus)
	reachable(Ctl_Monitor = sufficient)
	reachable(Ctl_Monitor = optimal)
	reachable(Ctl_Monitor = few)
	reachable(Ctl_Monitor = short)

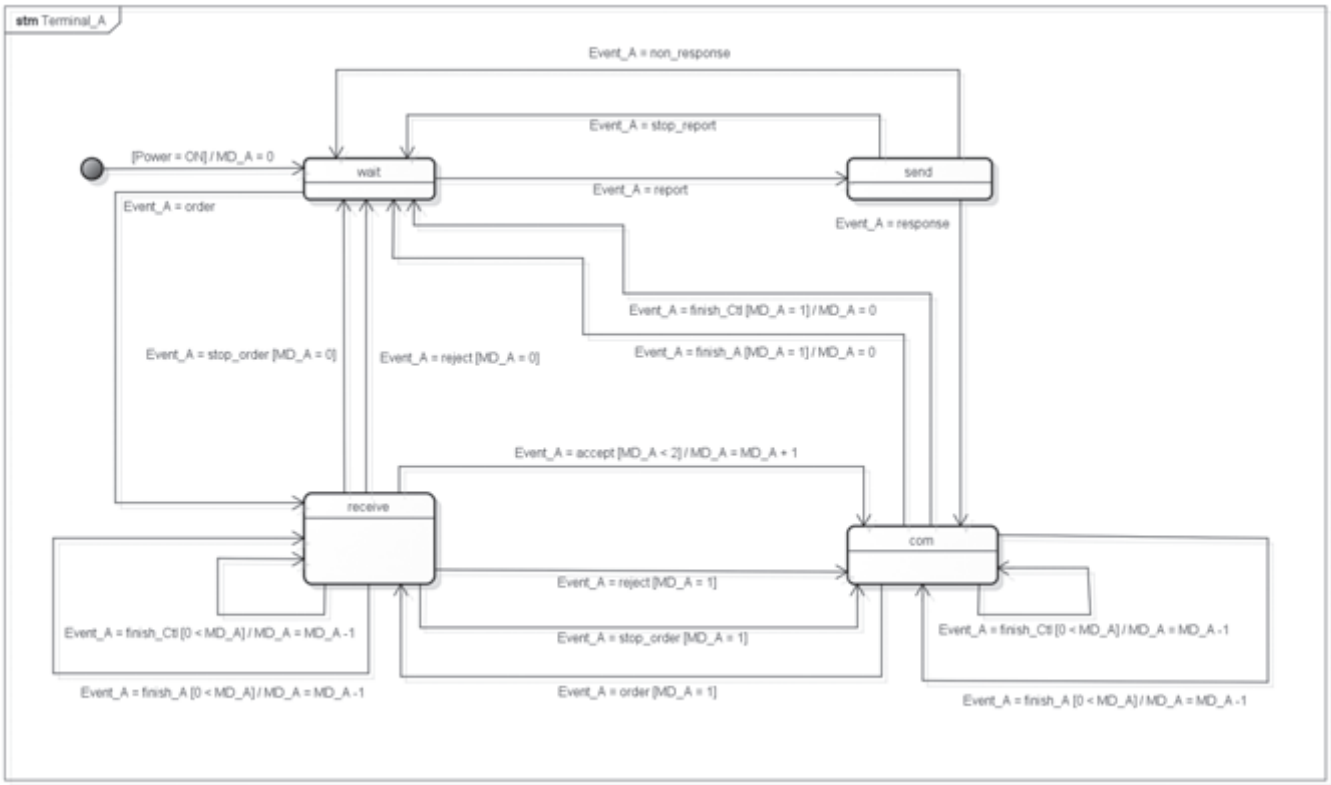

図４　端末 A の状態マシン図

業員が持つ端末は同時に 2 通話が可能である．
・管理者側の端末には，通信回線の利用数に応じて，従業員の配置数を評価する指標が過剰 (0) ～不足 (4) の 5 段階で表示される．

3.2. 例題システム

R-CDS の状態遷移表を表 3 に示す．ここでは，従業員は 2 名としそれぞれの持つ端末を端末 A および端末 B とする．表 3 の行はイベントを，列は端末 A の状態を表す. 各セルの上段は遷移の条件と遷移先を [条件 : 遷移先] の形で記述する．下段は遷移の際に実行されるアクションを表している． ここで MD_A は端末の通信モードを表しており，MD_A =0 の場合は通信しておらず，MD_A =1 および 2 の場合は 1 回線および 2 回線で通信していることを表している．端末 B の動作は，表 3 の変数 MD_A を MD_B で置き換えて得られる状態遷移表によって記述される．端末 A の状態遷移表から作成した状態マシン図を図 4 に示す．端末 B についても同様に状態マシン図を作成する．

R-CDS では，変数 MD_A および MD_B の値に応じて管理者側の端末に従業員の配置数を評価する指数が 5 段階で表示される．通信回線の利用数が多いことは，売り場への商品補充指示が多いことを表しており，従業員が不足することを意味する．逆に，通信回線の利用数が少ないことは，従業員が過剰であることを意味する．MD_A および MD_B の値に応じた管理者側の端末への表示の変化を表す状態マシン図を図 5 に示す．

本実験では，まず (1) 状態遷移表の各状態への到達可能性，(2) 各端末の通信量に関する到達可能性，(3) 各端末の通信量に関する安全性，(4) 管理者側の端末の表示に関する到達可能性，の 4 つの特性について検証を行った．これらの特性は，仕様テンプレートを用いて表 4 に示すように記述できる．

次に，事例提供元から要望があった 3 つの特性について検証を行った．検証した特性は，(1) 端末が待機状態 (wait) にもかかわらず通信中である状態への到達可能性，(2) 端末が 2 回線使用中にもかかわらず着信中 (receive) となる状態への到達可能性，(3) 端末が通話中 (com) にもかかわらず通話なしである状態への到達可能性，である．これらの特性は仕様テンプレートを用いず直接 CTL を用いて記述した．検証に用いた CTL 式を表 5 に示す．

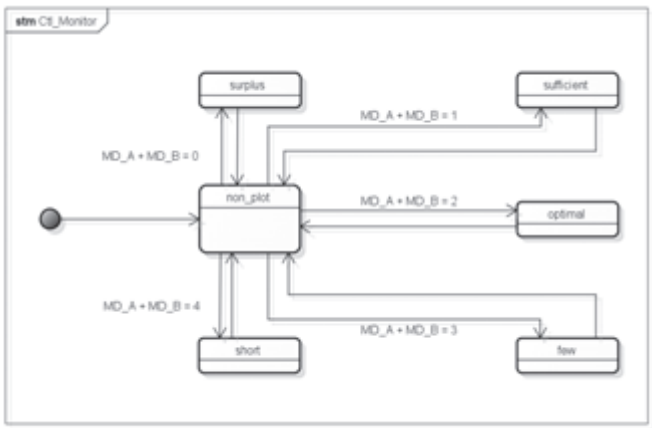

図 5　管理者端末への表示の状態マシン図

3.3. 結果

NuSMV による検証結果から得られたファイルを図 6 に示す．図 6(a) に示す通り，表 4 に示す特性はすべて TRUE となり，誤りがないことが確認できた．次に，図 6(b) に示す通り，表 5 の特性については，3-A および 3-B が FALSE となり，特性を満たさないという結果が得られた．図 6(c) の 3-A に対する反例を解析したところ，Terminal_A=com，すなわち端末 A が通信中となったにもかかわらず，MD_A の値が 0，すなわち通話なしの状態のままとなる系列が存在していた．Terminal_A の値が send から com となる遷移を確認したところ，表 3 の状態遷移表で記述されていた MD_A=1 のアクションが，図 4 の状態マシン図では記述されていなかった．そのため，Terminal_A=com となっても MD_A の値が 0 から変わらず，3-A の CTL 式に違反する状態へと到達していた．以上から，この誤りはユーザによる状態マシン図作成時のミスであることが判明した．

3.4. 評価

適用実験の結果をうけて，事例提供元からは，今回は転記ミスが原因ではあったものの，記述の漏れや間違いを発見できる技術として，モデル検査は非常に有用であるという評価を得た．さらに，違反が検出された際に，反例を用いて原因を特定できる点についても高く評価された．一方で，モデル検査の導入を容易にするという面から，本ツールの有効性についても一定の評価を得た．また，ツールの機能面に関して，状態マシン図の全ての記法に対応して欲しいとの要望をうけるともに，GUI の実装による利便性の向上についての要望があった．

表 5　事例提供元の要望に基づく検査特性

No	検査項目	CTL 式
1-A	端末が待機状態にもかかわらず通信中である状態への到達可能性	!EF (Terminal_A = wait & !(MD_A = 0))
1-B		!EF (Terminal_B = wait & !(MD_B = 0))
2-A	端末が 2 回線使用中にもかかわらず着信中となる状態への到達可能性	!EF (Terminal_A = receive & MD_A = 2)
2-B		!EF (Terminal_B = receive & MD_B = 2)
3-A	端末が通話中にもかかわらず通話なしである状態への到達可能性	!EF (Terminal_A = com & MD_A = 0)
3-B		!EF (Terminal_B = com & MD_B = 0)

4. 考察

4.1. 既存の技術との比較

UML の状態マシン図を対象としてモデル検査による設計検証を行うための研究開発は，これまでにも盛んに行われている [Bhaduri2004]．中でも，SPIN を用いたもの [Latella1999] と，本ツールと同様に SMV を用いたもの [Chan1998][Clarke2000] が著名である．

Latella ら [Latella1999] の提案した手法は，状態マシン図の構造を階層化オートマトンで表現した上で，それらを PROMELA 言語によるモデルへと変換し，モデル検査器 SPIN を用いて検証を行うものである．

一方で，Chan ら [Chan1998] の提案した手法では，状態マシン図と同様の構造および意味論をもつ設計記法である RSML を対象として，SMV による検証を行っている．モデル化においては，遷移による状態や変数の値の変化を本ツールと同じく case 文によって記述する．そして，Clarke ら [Clarke2000] の提案した手法では，状態マシン図の各遷移による動作を論理式として書き下し，結合することによって状態マシン図全体の振る舞いをモデル化している．

Latella らの手法では，複数の状態マシン図を用いて記述される振る舞いは考慮していない．本論文の事例のように，複数のコンポーネントが相互に通信を行うようなソフトウェアの設計を対象とする場合は，手法の拡張が必要となる．Chan らの手法では決定的動作のみを考慮している．すなわち，複数の遷移が同時に実行可能となり，非決定的にどちらかが選ばれるような振る舞いを

```
(001) EF Terminal_A = send  is true
(002) EF Terminal_A = receive  is true
(003) EF Terminal_A = com  is true
(004) EF Terminal_B = send  is true
(005) EF Terminal_B = receive  is true
(006) EF Terminal_B = com  is true
(007) EF MD_A = 1  is true
(008) EF MD_A = 2  is true
(009) EF MD_B = 1  is true
(010) EF MD_B = 2  is true
(011) AG !(MD_A = 3)  is true
(012) AG !(MD_A = -1)  is true
(013) AG !(MD_B = 3)  is true
(014) AG !(MD_B = -1)  is true
(015) EF Ctl_Monitor = surplus  is true
(016) EF Ctl_Monitor = sufficient  is true
(017) EF Ctl_Monitor = optimal  is true
(018) EF Ctl_Monitor = few  is true
(019) EF Ctl_Monitor = short  is true
```

(a) 表 4 の特性に対する検証結果
(result ファイル)

```
(001) !(EF (Terminal_A = wait & !(MD_A = 0)))
is true
(002) !(EF (Terminal_A = receive & MD_A = 2))
is true
(003) !(EF (Terminal_A = com & MD_A = 0))  is
false
(004) !(EF (Terminal_B = wait & !(MD_B = 0)))
is true
(005) !(EF (Terminal_B = receive & MD_B = 2))
is true
(006) !(EF (Terminal_B = com & MD_B = 0))  is
false
```

(b) 表 5 の特性に対する検証結果
(result ファイル)

```
(001) !(EF (Terminal_A = wait & !(MD_A = 0)))  is true
(002) !(EF (Terminal_A = receive & MD_A = 2))  is true
(003) !(EF (Terminal_A = com & MD_A = 0))  is false
-> State: 1.1 <-
  Terminal_A = initial
  MD_A = 0
  Power = start
  Event_A = emp
-> State: 1.2 <-
  Power = ON
  Event_A = report
-> State: 1.3 <-
  Terminal_A = wait
-> State: 1.4 <-
  Terminal_A = send
  Event_A = response
-> State: 1.5 <-
  Terminal_A = com
  Event_A = report
(004) !(EF (Terminal_B = wait & !(MD_B = 0)))  is true
(005) !(EF (Terminal_B = receive & MD_B = 2))  is true
(006) !(EF (Terminal_B = com & MD_B = 0))  is false
-> State: 2.1 <-
  Terminal_B = initial
  MD_B = 0
  Power = start
  Event_B = emp
-> State: 2.2 <-
  Power = ON
  Event_B = report
-> State: 2.3 <-
  Terminal_B = wait
-> State: 2.4 <-
  Terminal_B = send
  Event_B = response
-> State: 2.5 <-
  Terminal_B = com
  Event_B = report
```

(c) 表 5 の特性に対する反例
(ce ファイル)

図 6　NuSMV による検証結果から得られたファイル

もつ状態マシン図は扱うことができない．Clarke らの手法では，遷移に基づく振る舞いを全て論理式として表現している．そのため，case 文によるモデル化と比して探索対象となるモデルが複雑となり，検証コストが大きくなる可能性がある．また，このモデルでは COI(Cone of Influence) オプション [Berizin1998] による高速化の効果も得られ難い．

また，これらの手法では，検査対象となる仕様の入力支援についてはサポートしておらず，ユーザは時相論理を用いて直接仕様を記述する必要がある．そのため，専門知識をもたないユーザがこれらの手法を導入して設計検証を行うことは困難である．

このほかにも UML を対象とした設計検証手法は数多く報告されているが，その大部分は個別の例題への適用事例であり，汎用的に利用できるものではない．また，モデル化の手続きを一般化したものについても，実装したツールを現場で即座に利用できる形式で公開しているものは筆者の知る限りでは存在していない．

4.2. 産業界への展開

モデル検査技術はソフトウェアの設計を網羅的に検証し，設計の無矛盾性や仕様との整合性を確認するために有効である．また，モデル検査技術によるソフトウェア設計の検査について産業界からの期待も大きい．このような状況にも関わらず，組み込みソフトウェアの設計検証にモデル検査技術を導入した事例はまだまだ少ない．一部企業での導入事例は存在するが，産業界からの期待の大きさに比してモデル検査による設計検証を導入している企業の数は少ない．モデル検査を設計検証に導入している事例が少ない原因として，モデル作成の困難さが指摘されている．

この問題を解決するために，本ツールは，モデル検査による設計検証を行う上でのモデル作成を支援として，UML 図で記述されたソフトウェア製品の設計図を入力とし，モデル検査ツールの入力形式に自動で変換・出力できる．しかしながら，本ツールでは自動変換の効率化のために，UML 図の記法の多くの部分に制約をかけており，そのトレードオフとして利便性はある程度低下してしまう．適用事例における事例提供元との意見交換においても，状態マシン図のすべての記法に対応してほしいと要望を受けている．

また，本ツールの成果として，安全性・活性・到達可能性という登場頻度の極めて高い検査特性をテンプレートによって記述することが可能である．また，状態マシン図において重要な要素である状態・メッセージ・変数について扱うことが可能であることから，本研究で開発した仕様テンプレートは，ソフトウェアの代表的な特性を記述するにあたって十分な表現能力があると考えられる．しかしながら，より幅広い層のユーザへの普及のためにも，仕様パターンによる検査特性の記述への対応は取り組むべき課題であると考えられる．

最後に，本ツールでは現在のところコマンドラインでの実行のみが可能であり，GUI はサポートされていない．利便性を向上し，普及を進める上では GUI の実装が必要不可欠であると考えられる．

5. まとめ

本論文ではモデル検査によるソフトウェアの設計検証を支援するツールを開発した．協力企業で実際作成された設計文書へ本ツールを適用した結果，その有用性が確認できた．検証できる UML 図や性質の拡張およびツールのインタフェースの改良が今後の課題である．

謝辞

本研究は，独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター（SEC: Software Reliability Enhancement Center）が実施した「2013 年度ソフトウェア工学分野の先導的研究支援事業」の支援を受けたものです．

【参考文献】

[Clarke1999] E.M. Clarke, O. Grumberg and D. Peled: Model Checking, MIT Press (1999).

[McMillan1993] K. McMillan: Symbolic Model Checking, Kluwer Academic (1993).

[Holzmann1997] G. Holzmann: The Spin model Checker, IEEE Trans. Soft. Eng., 23(5), pp.279-295(1997).

[UML] O. M. Group: Unified Modeling Language Specification, Object Management Group (2001). http://www.uml.org.

[Burch1990] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang: Symbolic model checking: 10^{20} states and beyond, Proc. of the Fifth Annual IEEE Symp. on Logic in Computer Science (1990).

[NuSMV] NuSMV. http://nusmv.fbk.eu/.

[astah] astah* community http://astah.change- vision.com/.

[Tool] http://circuit.cse.oka-pu.ac.jp/tool.html/.

[Pnueli1977] A. Pnueli: The temporal logic of programs, Proc. of 18th IEEE Symp. Foundations of Computer Science (FOCS'77), pp.46-57 (1977).

[Clarke1981] E.M. Clarke and E.A. Emerson: Design and synthesis of synchronization skeletons using branching time temporal logic, Proc. of Logics of Programs Workshop, vol. 131 of Lecture Notes in Computer Science, pp.52-71 (1981).

[Bhaduri2004] P. Bhaduri and S. Ramesh: Model Checking of Statechart Models: Survey and Research Directions, ArXiv Computer Science e-prints (2004).

[Latella1999] D. Latella, I. Majzik, and M. Massink: Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker, Formal Aspects of Computing, 11(6), pp.637–664, (1999).

[Chan1998] W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin and J. D. Reese: Model checking large software specifications, IEEE Trans. Software Eng., 24, 7, pp. 498–520 (1998).

[Clarke2000] E. M. Clarke and W. Heinle: Modular translation of statecharts to SMV, Technical report, Carnegie-Mellon University School of Computer Science (2000).

[Berizin1998] S. Berezin, S.V.A. Campos, and E.M. Clarke: Compositional reasoning in model checking, Proc. of Int'l Symp. on Compositionality: The Significant Difference (COMPOS'97), pp.81–102, (1998).