社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2006-36 (2006-08)

UML/OCLに記述された時間QoSの階層的検証手法の提案

長井　栄吾† 　岡野　浩三† 　楠本　真二†

† 大阪大学大学院情報科学研究科
〒 560–8531 大阪府豊中市待兼山 1–3
E-mail: †{e-nagai,okano,kusumoto}@ist.osaka-u.ac.jp

あらまし　分散実時間アプリケーションにおいては，ユーザに対して提供される QoS (Quality of Service) を保証することがきわめて重要である．特に，時間 QoS に対する要求の保証が重要視されている．本稿では，UML/OCL に記述された時間 QoS を多段階かつ階層的に検証する方法を提案する．コンポーネントの時間的振る舞いを時間オートマントで抽象化した抽象 QoS モデルを利用することで，時間 QoS を多段階かつ階層的に形式的に保証する．これにより，状態爆発など既存の手法が抱える問題を解消し，効率的に時間 QoS の整合性の判定が可能となる．提案手法を UPPAAL を用いて例題に適用した結果，検証時間の観点から見て有用であることがわかった．

キーワード　分散実時間アプリケーション，時間 QoS，モデル検査，UPPAAL，形式的検証

Hierarchical Model Checking to Ensure Timeliness QoS Described in UML/OCL

Eigo NAGAI†, Kozo OKANO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University
Machikane-yama 1–3, Toyonaka City, Osaka, 560–8531 Japan
E-mail: †{e-nagai,okano,kusumoto}@ist.osaka-u.ac.jp

Abstract　Distributed Real-time application should generally ensure QoS (Quality of Service) providing to end-users. In particular, to ensure timeliness QoS is important. The paper proposes a hierarchical approach to verify formally timeliness QoS for UML/OCL specification of real-time applications. The method uses an abstract QoS model which models abstract behavior of a given component in a timed automaton. Also the method can be applied recursively along with depth of component hierarchy. Thus, the method avoids the state-explosion problem and verify effectively the timeliness QoS properties. Applying the proposed method to an example with UPPAAL, one of the famous model checkers, finds the method is useful.

Key words　Real-time Distributed System, Timeliness QoS, Model Checking, UPPAAL, Formal Verification

1. まえがき

近年，インターネットの爆発的普及と同時に，音声・動画を扱う分散実時間アプリケーション（マルチメディアシステム）に対して，高品質なサービスをユーザに提供することが求められおり，アプリケーション設計フェーズからQoSについて考慮に入れておくことが重要である．特に，時間に関するものは時間 QoS と呼ばれ，分散実時間アプリケーション開発においては考慮が必要である．本手法では，時間 QoS のうち，スループット (Throughput)，ジッタ (Jitter)，および遅延 (Latency) の 3 カテゴリを扱う．

一般的に，分散実時間アプリケーションは特定の機能を有したコンポーネントの集合として構成されるコンポーネントベースシステムであることが多い．また，そのようなアプリケーションの開発支援を目的とし，UML(Unified Modeling Language)[1] を代表とする様々なオブジェクト指向技術が提唱されている．UML には，モデルの各要素に対して制約を形式的に与えることができる OCL(Object Constraint Language) が標準装備されており，各コンポーネントに対する QoS を OCL を用いて記述することが可能である．

アプリケーションを構成する各コンポーネントやアプリ

ケーション全体の時間 QoS に関する検証手法については，テストオートマトンの概念に基づいた形式的アプローチ [2] が存在する．本手法では，時間オートマトンによるモデル検査ツール UPPAAL [3] を用いて，テストオートマトンを動作させることにより，時間 QoS が満たされているかを判定する．

本稿は我々の以前の論文 [9] において課題と考えられた点を改良したものである．以前の手法はコンポーネント間の検証手法において線形計画法を用いている．そのため，課題点としてコンポーネントの接続関係に閉路が存在してはならない点，原則としてコンポーネントが多段階にわたって階層構造をなす場合に適用が困難だった点が挙げられる．提案手法では，線形計画法に替わり，各コンポーネントの時間 QoS の側面を抽象オートマトンで簡潔に抽象化し，モデル検査ツール UPPAAL を利用したモデル検査を行うことで，上記の課題を克服することを目的としている．

本稿で提案する手法は多段階階かつ階層的に時間 QoS を検証できる．階層的にモデル検査を行うことにより，従来問題とされてきた状態爆発を回避しやすくなり，同時に多段階かつ階層的な UML/OCL によるコンポーネントベースの設計仕様に対応することができる．本提案手法では，2 段階の検証を行う．まず第 1 段階として，コンポーネントの提供する時間 QoS がアプリケーション全体の必要とする QoS を満たすかを判定する．UML/OCL に記述されたコンポーネント間の時間 QoS を抽象 QoS モデルとして時間オートマトン化し，そのテストオートマトンを動作させることで検証する．次に第 2 段階として，コンポーネントの提供する時間 QoS が，そのコンポーネントの動作記述上で保証されるかを判定する．コンポーネント内部のより詳細な動作記述であるステートチャート図を等価な時間オートマトンへと変換し，テストオートマトンと並行動作させることで検証を行う．一般の設計では，各コンポーネントごと必要なら上記のプロセスを繰返す．

以降，2. では準備として時間 QoS について簡単に説明する．次に 3. で，UML/OCL における時間 QoS の記述について説明する．4. ではコンポーネント間の時間 QoS 検証手法について述べ，5. にてコンポーネント内部の詳細な動作記述に対する時間 QoS 検証手法について述べる．6. において提案手法を例題に適用する．最後に，7. でまとめる．

2. 時 間 QoS

分散実時間アプリケーションはコンポーネントベースシステムであることが多い．各コンポーネントは独立したインターフェースを持っており，コンポーネント間の相互作用は全てインターフェースを通して行われる．従って，コンポーネントのインターフェースに指定された入出力信号のスループット，ジッタ，および遅延に着目することで，時間 QoS を表現することができる．

ジッタ，スループット，および遅延の時間 QoS は信号発生順をもとに以下のように定型化される．入出力信号を x, y，および z とし，信号の発生順序を添え字として表す．つまり，信号 x はその発生順に $x_1, x_2, ...$ となる．では，ジッタ，スループット，および遅延について，それぞれ説明する．

スループット 時間 T 間において，K 回の信号 x を発生する場合，次のような線形制約式となる．

$$\forall i \in \mathbb{N} : x_{i+K-1} - x_i \leqq T (\forall i \in \mathbb{N} : x_{i+K-1} - x_i \geqq T)$$

ジッタ 時間 T 間において，ジッタが m, M で信号 x を発生する場合，次のようになる．

$$\forall i \in \mathbb{N} : T - m \leqq x_{i+1} - xi \leqq T + M$$

遅延 信号 x, y が時間 T の遅延である場合，

$$\forall i \in \mathbb{N} : 0 \leqq x_i - y_{K_i+K'} \leqq T$$

となり，特に $K = 1, K' = 0$ においては，i 番目の x, y の遅延となる．

$$\forall i \in \mathbb{N} : 0 \leqq x_i - y_i \leqq T$$

3. UML/OCL による時間 QoS 記述

UML にはモデルの各要素に形式的制約を与える OCL が標準装備されており，それを用いた QoS などの記述が考案されている [9]．本章では，検証の入力として必要なコンポーネント間の接続関係，QoS 記述について簡単に説明する．

3.1 コンポーネント間接続関係の記述

本手法では，コンポーネント間の接続関係を図 1 のような UML クラス図を用いて記述する．

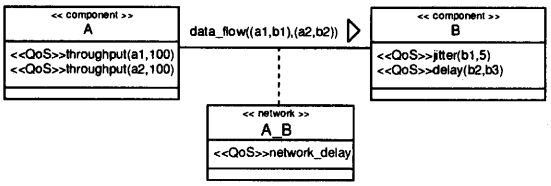

図 1 クラス図によるコンポーネント間接続関係

コンポーネントは “component” ステレオタイプの付いたクラスを用いて指定する．ステレオタイプ (Stereotype) とは UML の拡張を実現する機構の一つであり [1]，モデルの各要素に対してユーザ定義を与えることが可能となる．

コンポーネント接続関係は UML クラス図の関連 (Assosiation) をもって，それらのクラスに対応するコンポーネント同士が接続関係にあることを記述する．本手法で定義する「接続関係」は，何らかのデータ通信がなされるコンポーネントの組の間で規定される．関連名として以下の形式を採る．

関連名 := “data_flow(” データ送受信群 “)” ;
データ送受信群 := データ送受信群 * ;
データ送受信 := “(” 送信側データ名 “,” 受信側データ名 “)” ;

例えば，図 1 ではコンポーネント A からデータ a1，a2 がコンポーネント B に送信され，コンポーネント B はそれらをデータ b1，b2 で受信していることが関連名により，明示されている．

3.2 コンポーネントおよびネットワークに対する時間 QoS の記述

次に，各コンポーネントおよびネットワークに対して時間 QoS を課す．本手法では，各コンポーネント ("component" ステレオタイプの付いたクラス) およびネットワーク ("network" ステレオタイプの付いた関連クラス) に対して QoS 用の変数を保持させ，変数に対する制約として OCL で時間 QoS を記述するという方針をとる．以下，記述方法を説明する．

まず，図 1 のように，各クラスの属性区画に "QoS" ステレオタイプのついた QoS 用の変数を用意する．本手法においては，スループット，ジッタ，遅延の 3 カテゴリを扱うため，変数の形式は以下の形式とする．

スループット変数 := "throughput(" データ名 "," 期間 ")";
ジッタ変数 := "jitter(" データ名 "," 期間 ")" ;
処理遅延 := "delay(" 送信データ "," 受信データ ")";
通信遅延 := "network_delay" ;

"component" ステレオタイプの付いたクラスについては，スループット，ジッタ，処理遅延の 3 カテゴリ，"network" ステレオタイプの付いた関連クラスについては通信遅延の 1 カテゴリを変数として指定できる．次に，OCL で時間 QoS を記述する．1 クラス，つまり，1 コンポーネントあるいは 1 ネットワークに対する時間 QoS を以下の形式で記述する．

QoS 記述 := "context" クラス名 不変式* ;
不変式 := "inv: self." 制約式 ;
制約式 := 変数 演算子 正整数;
変数 := スループット変数 | ジッタ変数 | 処理遅延 | 通信遅延;
演算子 := ">" | "<" | "≧" | "≦" ;

4. コンポーネントベースの検証法第一段階

本章では，コンポーネント間の接続関係と各コンポーネントの提供する時間 QoS (以降，提供 QoS) からアプリケーション全体に求められる QoS (以降，要求 QoS) が満たされるかを検証する手法について説明する．検証においては以下のパラメータを必要とする．

- 要求 QoS
- 各コンポーネントの提供 QoS
- コンポーネント間の接続を表すオートマトン (以降，コンフィギュレーションオートマトンと呼ぶ)

上記のパラメータを入力として与え，提供 QoS から抽象 QoS モデルを生成し，テストオートマトンの概念を利用して，検証を行う．

4.1 モデリング

従来の手法 [5] では，開発者はネットワーク上の全てのコンポーネントの振る舞いを一度に時間オートマトンにモデル化し，モデル検査を行っていた．しかし，これでは状態爆発を行う可能性が極めて高い．

そこで，本手法では各コンポーネントの提供 QoS の動作を示す時間オートマトン (以降，抽象時間 QoS オートマトンとする) を，以下に提案する変換ルールを適用し，生成する．生成された抽象時間 QoS オートマトンは抽象モデルとして，UPPAAL により，モデル検査される．得られた時間オートマトンは比較的小さいので，状態爆発の危険が減少するといえる．

次節にて，各提供 QoS からの抽象時間 QoS オートマトンへの変換ルールを述べる．

4.2 変換ルール

スループット 時間 T 間における最大信号発生回数を M，最小を m，出力信号を x，およびスループットバッファ変数 c とした場合，変換されるオートマトンは図 2 となる．

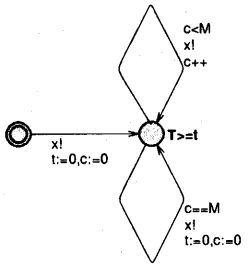

図 2 スループットの抽象時間 QoS オートマトン

ジッタ 時間 T 間におけるジッタの揺らぎを $+d1$，最小を $-d0$，および出力信号 x とした場合，変換されるオートマトンは図 3 となる．

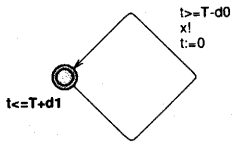

図 3 ジッタの抽象時間 QoS オートマトン

遅延 最大遅延を M，最小遅延を m とすると，変換されるオートマトンは図 4 となる．ただし，遅延については入力 y を受信するまでに，出力 x が要求された場合，同様のオートマトンが必要となる．必要な数はコンポーネントのスループット特性により決定する．

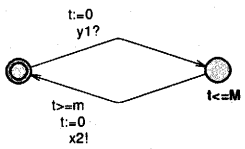

図 4 遅延の抽象時間 QoS オートマトン

4.3 コンフィギュレーションオートマトン

コンフィギュレーションオートマトンは各コンポーネントのインターフェースを表現した時間オートマトンである．各コンポーネントはインターフェースに指定された入出力信号を持っているため，それを UPPAAL 拡張時間オートマトンにおける *channel* として与える．*channel* は，コンポーネントの提供 QoS の変換オートマトンと同期する．このようにして，提供 QoS による変換オートマトンとコンフィギュレーションオートマトンが同期をとる．

4.4 状態数減少の工夫

ほとんどのアプリケーションでは，ネットワーク特性として主に遅延のみがとりあげられる．このような場合，コンフィギュレーションオートマトンにおける該当ロケーションの *invariant* として遅延を記述することで，状態数を減少させることが可能である．

また要求 QoS がジッタ，スループットおよび遅延の全てに求められた場合，QoS をカテゴリ別に分類しモデル検査を行う方法により，状態数を減少させることが可能である．第一段階として，ジッタと遅延のみの抽象モデルを生成し，それをモデル検査する．第二段階はスループットと遅延についての抽象モデルに対して同様にモデル検査を行う．こうすることで，状態数が減少し，状態爆発の可能性を減らすことができる．

4.5 検証手法

検証方法は，テストオートマトンの原理にもとづいて，次のようになる．次にシステムに与えられた要求 QoS をテストするためのテストオートマトンと各コンポーネントの提供 QoS から得られた抽象オートマトン，および全体構成を表すオートマトンを並列に動作させる．その際，入力された時間オートマトンネットワークがデッドロックを発生させないか (検証式は "A[] not deadlock") を UPPAAL で検証する．デッドロックが起こった場合，与えられた時間 QoS を満たしていないことがわかる．

5. コンポーネントベース検証法第二段階

本章では各コンポーネントごとに，その動作記述が QoS を満たすかを単体テストする手法について説明する．これは基本的には文献 [9] の内部設計検証と同じ方法である．

5.1 モデリング

設計者は各コンポーネントごと内部の動作の詳細設計を (再び下位のコンポーネントを必要なら組合わせて階層的に) 行う．この場合の検証法はもとのコンポーネントの提供 QoS を要求 QoS と読み替えて，再び 4.5 の方法で検証を行う．

ここでは下位のコンポーネントを用いず，内部動作をすべてステートチャートで表せたとして説明する．

UPPAAL への入力を生成するため，UML ステートチャートを時間オートマトンに変換する必要がある．ステートチャートは階層的構造が記述できるモデルである一方，UPPAAL 時間オートマトンは平面的 (Flat) な記述モデルである．一般的に，階層構造を平面モデルに変換すると，状態数は増加する．階層構造を保ったまま変換を行う手法も存在する [7] が，設計者は各コンポーネントごと内部の動作の細かな流れを（再び詳細なコンポーネントを組あわせて）設計する。この場合の検証法は前節と同じとなる。変換後の構造が複雑となり，動作が複雑になるという問題があるため，時間 QoS の要求が強い実時間システムに対して階層構造を保ったまま変換する手法を用いることは好ましくない．したがって，本手法では階層構造を平面モデルに変換するアルゴリズム [8] を採用する．ただし，この変換手法は，階層時間オートマトン (Hierarchical Timed Automata) を UPPAAL 時間オートマトンへ変換するアルゴリズムであるため，本手法では階層時間オートマトンを経た 2 段階変換を行う．UML ステートチャートと階層時間オートマトンは階層構造であるという類似点を持ち，記述方法も似ているため，2 モデル間の変換は簡単な文法変換に帰着できる．

5.2 テストオートマトン

ジッタはスループットの一種であるため，ジッタとスループットをテストするオートマトンは同じものとなる [5]．データ *in* がある一定区間 $RATE$ 毎に発生するという条件のもとで，(期間/スループット) あるいは (ジッタ) の最小値 MIN，最大値 MAX を満たしているかを確認しながら状態遷移が行われる．RateTooSlow あるいは RateTooFast の状態に到達するとデッドロックが発生し，与えられたジッタあるいはスループットを満たしていないことがわかる．なお，文献 [2] にも同様のテストオートマトンが 3 種 (Anchored, Non-Anchored の区別も含めて)，提案されている．

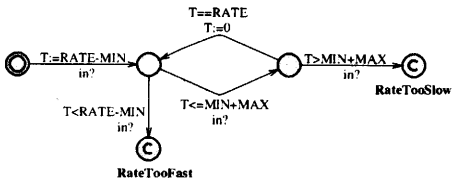

図 5　ジッタ，スループットのテスト時間オートマトン

また，遅延のテストオートマトンに関しては，データ *start* が発生してからデータ *check* が発生するまでの遅延時間 *latency* を確認しながら状態遷移が行われている．データは連続して発生するためデータ *start* が発生して *check* が発生するまでの間に次のデータ *start* が発生することもある．各々のデータ *start* の発生時間を記録するためにクロックは複数用意せねばなず，図 6 のテストオートマトンではクロック変数が配列として宣言されている．LatencyFailed あるいは QueueTooSmall の状態へ到達するとデッドロックが発生し，与えられた遅延を満たしていないことがわかる．

5.3 検証手法

検証方法は，4.5 節で述べたものと同じである．

6. 例題適用

本提案手法による検証を例題に適用した．なお，実験

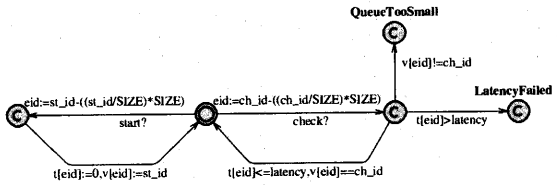

図6　遅延のテスト時間オートマトン

環境は OS が Microsoft Windows XP Professional，CPU が PentiumIV 2.6GHz およびメモリが 1 GB である．

6.1　例題概要

メディアサーバ (Media Server) は，ユーザが指定した映像データあるいは音声データを，デジタルテレビ (Digital Television) あるいはオーディオシステム (Audio System) に随時配信するアプリケーションである [5]．各出力機器には指定された時間 QoS(出力スループット) を保証することが強く要求されるため，メディアサーバを例題とし，提案手法を適用した．アプリケーション全体のコンポーネントの接続関係を記述した UML クラス図の概要を，図 7 に示す．計 12 個のコンポーネントから構成されている．また，メディアサーバ・デジタルテレビ間およびメディアサーバ・オーディオシステム間の接続関係については，ネットワークによるデータ送受信の関係を明示するために，関連クラスが用いられている．

6.2　検証第一段階

例題の提供 QoS の一部として以下を課す．

- コンポーネント MS-Server の出力スループットは 100 frames/s 以上である．
- コンポーネント MS-Storage の処理遅延 5 ms 以下である．
- メディアサーバ-デジタルテレビ間のネットワーク遅延は 100ms 以下である．
- メディアサーバ-オーディオシステム間のネットワーク遅延は 150ms 以下である．

これらの提供 QoS と UML クラス図で記述されたコンポーネント接続関係，さらにアプリケーションに要求されている時間 QoS を入力とし，アプリケーション全体の時間 QoS の整合性に関して，4. で述べた検証を行う．ここでは，要求されている要求 QoS として，“デジタルテレビのディスプレー (DT Display) の出力スループットは 30 franes/s 以上である” が挙げられているとし，検証を行った．検証におけるコンフィギュレーションオートマトンを図 8 に示す．

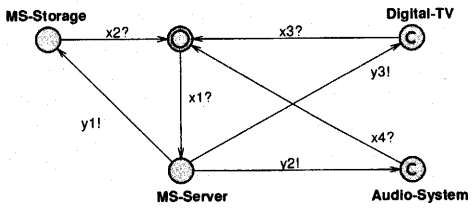

図8　コンフィギュレーションオートマトン

図 9 はコンポーネント MS-Storage の時間オートマトンモデルであり，入力信号は $y1$，出力信号 $x2$，処理遅延 M となる．図 9 は 4.2 に従って生成される．またコンポーネント MS-Storage の処理遅延 5 ms 以下という提供 QoS より $M = 5$ である．

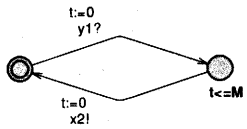

図9　コンポーネント MS-Storage の抽象時間 QoS オートマトン

また本例題と同じ検証を線形計画問題の解法パッケージを利用する手法 [9] との検証時間の比較を表 1 にまとめた．

表1　検証時間

既存手法 [5]	測定不能
線形計画法	76 ms
提案手法	150 ms

6.3　検証法第二段階

6.2 において，アプリケーション全体の整合性を確認した後，各コンポーネントの動作仕様を設計する．

動作仕様は UML ステートチャートで記述される．このとき，6.2 で与えられた時間 QoS を満たすように記述する必要がある．例えば，図 10 はコンポーネント MS-Storage の動作仕様である．

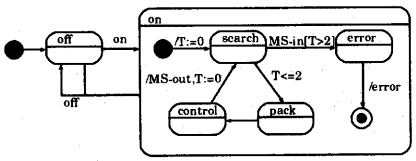

図10　コンポーネント MS-Storage のステートチャート

各コンポーネントに関する時間 QoS の検証のために，ステートチャートを UPPAAL 時間オートマトンネットワークへ，OCL で記述された時間 QoS をテストオートマトンへ自動変換する．図 11 は，図 10 のステート *on* 内部を変換した結果である．以下は，本例題における変換時間等の結果である．

変換時間　　: 1153ms
状態数 (変換前)　: 89 個
状態数 (変換後)　: 179 個

次に，各コンポーネントに対応する時間オートマトンを単体で動作させることで，システムそのものの誤りによるデッドロックを起こさないかどうかを検査する．最後に，各コンポーネントごとに，ステートチャートの変換結果である時間オートマトンとテストオートマトンを並列に動作させ，デッドロックが発生しないかを検査する．後者においてデッドロックが発生した場合は，テストオートマトン

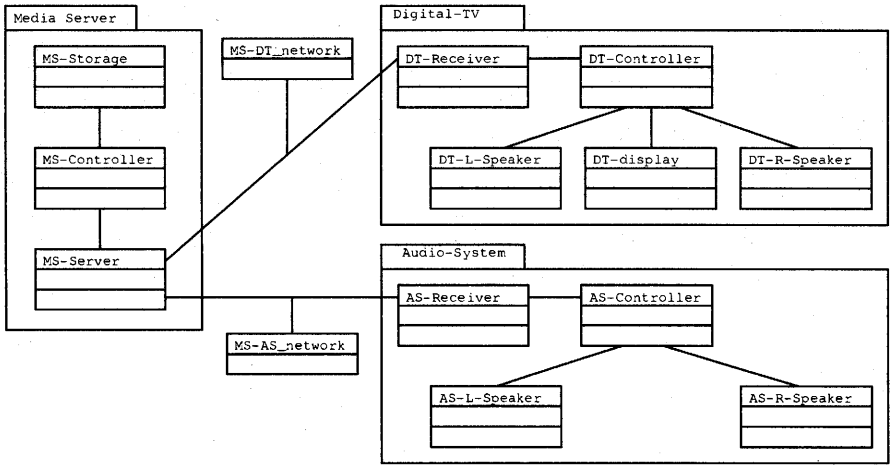

図 7　メディアサーバアプリケーションのクラス図

Fig. 7　A UML Class Diagram of Media Server Application

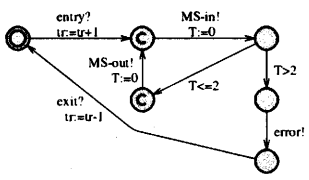

図 11　コンポーネント MS-Storage の UPPAAL 時間オートマトン

の規定する時間制約に反することを意味する．

結果，本例題においては，状態爆発を起こすこともなく，全てのコンポーネントに対して数秒以内に UPPAAL を用いて検査を行うことができた．

6.4 考　　察

6.2 では，線形計画問題の解法パッケージを利用した手法よりも，検証時間において提案手法は劣っているものの，十分に有用な範囲であると考えられる．ただし，本例題では提供 QoS が少ないという点があり，提供 QoS が増加すればするほど，状態数の増加による状態爆発の危険性が高まる．

しかしながら，線形計画法による手法はコンポーネント接続関係において閉路はないことが必要条件であるのに対し，提案手法では閉路があっても問題がないといった利点がある．また，提案手法はコンポーネントの構成について制約が線形計画法による手法にくらべて緩いため，多段階にわたるコンポーネントの詳細化が行えるようになる．

7. おわりに

本論文では，UML/OCL に記述された時間 QoS をコンポーネント間，コンポーネント内部の 2 つの階層に分けてモデル検査を行った．階層的に検証を行うことで，状態爆発問題の危険性を低下させることが可能となる．これらの検証により，コンポーネント同士の入出力インターフェースが複雑に絡み合うようなアプリケーションにおいても，時間 QoS の保証を自動的に行うことが可能となると考えられる．

今後の課題として，提案手法では複数の QoS を組み合わせた抽象オートマトンに対する検証に対応していない．これに対応する検証法を考案したい．また検証のトレースを解析することによりフィードバックを開発者に与えるようにすることが考えられる．

文　　献

[1] Object Management Group: Unified Modeing Language Specification version 1.5, available at http://www.omg.org/.

[2] Behzad Bordbar and Kozo Okano: “Verification of Timeliness QoS Properties in Multimedia Systems,” In Proceedings of the 5th International Conference on Formal Engineering Method (ICFEM ’03), Lecture Notes in Computer Science, Vol.2885, pp.523.540, 2003.

[3] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi: “UPPAAL . a Tool Suite for Automatic Verification of Real-Time Systems,” In Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, Lecture Notes in Computer Science, Vol.1066, pp.232.243, 1995.

[4] UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Request for Proposal, available at http://www.omg.org.

[5] D. Akehurst, J. Derrick, and A. G. Waters: “Design and Verification of Distributed Multi-media Systems,” LNCS, Vol. 2884, pp.276.292, 2003.

[6] K. Havelund, A. Skou, K. G. Larsen, and K. Lund: “Formal Modeling and Analysis of an Audio/Video Protocol: An Industrial Case St udy Using UPPAAL,” BRICS Technical Report Series, RS-97-31, 1997.

[7] A. Wasowski: “On Efficient Program Synthesis from Statecharts,” In Proc. of the 2003 ACM SIGPLAN Conf. on Language, Compiler, and Tool for Embedded Systems, Vol.38(7), pp.163–170, 2003.

[8] A. David and M. O. Möller: “From HUPPAAL to UPPAAL: Translation from Hierarchical Timed Automata to Flat Timed Automata,” BRICS Technical Report Series, RS-01-11, 2001.

[9] 長井 栄吾, 牧寺 彩, 岡野 浩三, 谷口 健一: “時間制約を保証する UML/OCL を用いた分散実時間アプリケーション開発手法,” 電子情報通信学会論文誌, Vol. J89-D No.4, pp683-692, 2006.