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

長井　栄吾[†]　　牧寺　彩[†]　　岡野　浩三[†]　　谷口　健一[†]

A Method to Develop Distributed Real-Time Applications Based on UML/OCL

Eigo NAGAI[†], Aya MAKIDERA[†], Kozo OKANO[†], and Kenichi TANIGUCHI[†]

あらまし　本論文では，UML/OCL を用いた分散環境実時間アプリケーション開発を支援する手法を提案する．提案手法により，UML/OCL を用いた開発アプリケーションの設計記述に対する，時間オートマトンを用いた timeliness QoS の効率良い整合性検証，時間制御コード自動生成が可能となる．設計，検証作業は 2 段階に分けられ，これにより検証の効率化が図られる．本論文では，提案手法に基づいた検証系及び導出系の実装，更に例題に対する適用結果について述べる．

キーワード　UML/OCL，実時間アプリケーション，Timeliness QoS，時間オートマトン，Java

1. まえがき

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) を保証することが極めて重要である．特に，timeliness QoS に対する要求は非常に高い [3]．そこで，本論文では timeliness QoS に着目した分散実時間アプリケーション開発手法を提案する．

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ことが多い [4]．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 [5] 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) [6] 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 を用いて記述できる．

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 timeliness 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が存在する [7], [8]．これらのアプローチでは，アプリケーション全体の動作振舞いが拡張時間オートマトンネットワークでモデル化され，CTL を用いて検証が行われる．ただし，コンポーネント数が多くなったりコンポーネント間の接続関係が複雑になるにつれ，メモリ使用量の面で検証が困難になるという問題をもつ．

本論文の提案する『分散実時間アプリケーション開発手法』とは，コンポーネントがネットワーク上に配置されたアプリケーションの UML/OCL を用いた設計記述から，効率の良い時間制約の整合性検証と整合性を満たす時間制御コード自動生成を行うものである．

本提案手法では，まず，アプリケーションの全体設計として，各コンポーネントが提供する Timeliness QoS は OCL を，コンポーネント間の接続関係は UML クラス図を用いて与える．更に，与えられた Timeliness QoS 集合とコンポーネント接続関係の条件下においてアプリケーション全体として要求される Timeliness QoS が満たされるかどうかを検証する．検証問題は，OCL で記述された Timeliness QoS を時間変数を用いた線形制約式に変換し，それら線形制約式の集合に対する非可解性判定問題に帰着して判定する [14]．

分散実時間アプリケーション設計及び開発は本提案手法を用いた以下の手順に従って行える．

（1）　アプリケーションの外部設計として，各コンポーネントが提供する timeliness QoS（以降，提供 QoS）は OCL を，コンポーネント間の接続関係

[†] 大阪大学大学院情報科学研究科，豊中市
Graduate School of Information Science and Technology, Osaka University, Toyonaka-shi, 560–8531 Japan

は UML クラス図を用いて与える．更に，与えられた timeliness QoS 集合とコンポーネント接続関係の条件下においてアプリケーション全体として要求される timeliness QoS（以降，要求 QoS）が満たされるかどうかを検証する．検証問題は，OCL で記述された timeliness QoS を時間変数を用いた線形制約式に変換し，それら線形制約式の集合に対する非可解性判定問題に帰着して判定する [14]．このアプローチの利点は，n 個のコンポーネントから構成されるアプリケーションの QoS 側面を $O(n)$ 個の線形制約式として抽出することにより，既存の手法のメモリ爆発問題を改善できることにある．

（2） 内部設計として，各コンポーネントの動作振舞い仕様を，クロックを付加した拡張 UML ステートチャートを用いて記述する．同時に，先に与えた各コンポーネントの提供する timeliness QoS を満たしているかを，記述した UML ステートチャートを等価変換した時間オートマトンとテストオートマトンを並行動作させることにより検証する．コンポーネント間の接続関係を考慮した検証は全体設計の (1) で済んでいるため，ここでの検証は各コンポーネントごとの単体検証を行えばよい．

（3） 次はアプリケーションの実装を行う．このとき，各コンポーネントは (1)，(2) で与えられた timeliness QoS を満たすようにプログラムコードを記述する必要がある．ここで，コンポーネントの timeliness QoS 制御部とアプリケーションの機能処理部の両方を記述する作業は非常に煩雑である．また，実時間処理を扱うアプリケーションでは，複数の動作を並行して実行できる機構や，ある機能処理を指定した時間内に完了できる機構が必要となる一方，プログラムコード自動生成は開発者が手作業でプログラムコードを記述するよりも開発生産性の向上を見込むことができ，開発されるアプリケーションの品質としても一定以上のものが期待できる．そこで提案手法では，timeliness QoS の保証された動作仕様記述からそれらと等価な状態遷移を行う Java プログラムコードを自動生成する．

このとき生成されるプログラムコードは，アプリケーションを構成する各コンポーネントの時間制御及びその時間制御を各コンポーネントが自律的に行うことを実現し，実行プログラムが異なる複数のクロックをもつ分散環境に配置された場合でも，全体として動作仕様記述どおりに時間制御を行う．

本論文では，提案手法に基づいた設計ツールのプロトタイプの実装を行い，例題メディアサーバに適用した．既存の分散アプリケーションの設計手法 [7] に対して，手順 (1) 及び手順 (2) のようにシステム全体の Timeliness QoS をコンポーネントレベルの Timeliness QoS に分割することで検証段階における負荷を軽減している．既存の手法では，状態爆発が起こり検証を完了させることができない例に対し検証を行うことができる．結果，検証やプログラムコード自動生成は数秒以内に結果を出力することが確認でき，時間の観点から見て有用であることが分かった．

以降，**2.** ではモデルチェックに使用するツール UPPAAL [2] の時相論理 CTL を説明し，**3.** では開発する実時間アプリケーション全体の外部設計と検証，**4.** では内部設計と検証，すなわち，UML ステートチャートから時間オートマトンへの変換系及び各コンポーネントの提供 QoS 整合性検証系，**5.** ではアプリケーションの実装としての時間制御コード生成系について述べる．**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. CTL

モデルチェックとは与えられた状態遷移系（モデル）がチェックしたい時相論理式を満たすかを判別することである．UPPAAL では時相論理として CTL (Computation Tree Logic) のサブクラスが使われる．

分岐時間時相論理である CTL は経路限定子（path quantifier）と時相演算子（temporal operator）の組合せによって時相を表現する．経路限定子には A，E の 2 種類がある．

以下に経路限定子 A，E と時相演算子 F，G，X，U は状態式 ϕ，経路式 ψ としたとき，次のような意味をもつ．

$A\psi$：これ以降のすべての経路で ψ を満たせば真

$E\psi$：これ以降に ψ を満たす経路が存在すれば真

$F\phi$：この経路上でいつか ϕ を満たせば真

$G\phi$：この経路上で常に ϕ を満たせば真

$X\phi$：この経路上の次の時間に ϕ を満たせば真

$\phi_1 U \phi_2$：この経路上でいつか ϕ_2 を満たし，かつそれまで ϕ_1 を満たし続ければ真

この経路限定子と時相演算子によって CTL の構文は次のように表せる [14].

$$\phi ::= p|\neg\phi|\phi_1 \vee \phi_2|EX\phi|E(\phi_1 U \phi_2)|EF\phi|EG\phi$$
$$|AX\phi|A(\phi_1 U \phi_2)|AF\phi|AG\phi$$

ただし p は原始論理式である

3. 外部設計検証

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．

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 [5]，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とる．

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 コンポーネント及びネットワークに対する Timeliness QoS の記述

次に，各コンポーネント及びネットワークに対して timeliness 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）及びネットワーク（“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 timeliness 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” ステレオタイプの付いた 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 カテゴリーを扱うため，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 カテゴリー，“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 カテゴリーを変数として指定できる．次に，OCL で timeliness 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 timeliness 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：= “>” | “<” | “≥” | “≤” ;

例えば，図 1 の 2 コンポーネント及び 1 ネットワークに対する timeliness QoS を OCL で与えると，以下のようになる．“--” 以降はコメントである．

```
context A
  inv: self.throughput(a1,100) ≥ 20
  - - データ a1 を期間 100 ms の間に送信する回数は 20 回以下
context B
  inv: self.jitter(b1, 5) < 1
  - - 5 ms 間隔で受信するとき，ジッタは 1 ms 未満
  inv: self.delay(b2,b3) < 5
  - - データ b2 を処理して b3 に渡すまでの遅延時間は 5 ms 未満
context A_B
  inv: network_delay≤ 100
  - - コンポーネント A とコンポーネント B の間のネットワーク遅延は 100 ms 以下
```

OCL による QoS 制約を線形制約で表現すると以下のようになる [8].

ある期間 T 内に信号 $\mathbf{x}$ が少なくとも K 回発生しなければならないという制約は次のように表現できる．

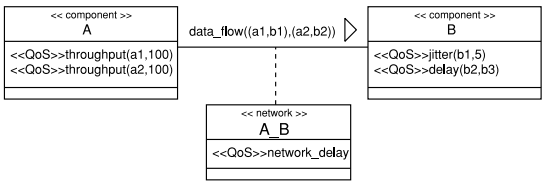

図 1 クラス図によるコンポーネント間接続関係
Fig. 1 A configuration of components in UML class diagram.

$\forall i \in \mathbb{N} : x_{i+K-1} - x_i \leq T$

同様に，ある期間 T 内に信号 $\mathbf{x}$ が K 回未満発生しなければならないという制約は次のように表現できる．

$\forall i \in \mathbb{N} : x_{i+K-1} - x_i \geq T$

間隔 T で発生する信号 $\mathbf{x}$ のジッタ制約は次のように表現できる．

$\forall i \in \mathbb{N} : T - m \leq x_{i+1} - x_i \leq T + M$ (m, M : 定数)

二つの信号 $\mathbf{x}$，$\mathbf{y}$ の遅延関係が T 未満であるという制約は次のように表現できる．

$\forall i \in \mathbb{N} : 0 < x_i - y_{i+C} \leq T$

OCL 制約式と線形制約式との対応関係は例えば次のようになる．

context A
inv: self.throughput(a1,100) ≥20
- - データ a1 を期間 100 ms の間に送信する回数は 20 回以下

となる OCL に対しては次のような線形制約式となる．

$\forall i \in \mathbb{N} : a_{i+20-1} - a_i \leq 100$

3.3 線形計画法を用いた整合性検証手法

ここでは，提供 QoS と要求 QoS の整合性を線形計画法を用いて調べる方法を述べる．整合性とは，各コンポーネントの提供 QoS とコンポーネントの接続関係のもとで要求 QoS が満たされることをいう．

これらの制約式は線形制約集合に変換され，制約領域の有無の判定問題を解くことにより [14]，提供 QoS が要求 QoS を満たすかを解く．

一般に，QoS 制約式は全称子を含むため線形制約式が無限個になり得るが，対象とする QoS のクラスを上記に制限し，コンポーネントに接続関係の制約を次のようにおくと，線形不等式の数は有限に抑えられる [13]．

整合性判定の制限及び，システムの制限として，以下の四つを課す．

[クラス制限]

（1） 要求 QoS にジッタ制約が含まれるとき，どこかのコンポーネントでジッタ制約が書かれていなければならない．

（2） 要求 QoS にスループット制約が含まれるとき，どこかのコンポーネントでスループット制約かジッタ制約が書かれていなければならない．また，要求スループット制約における定数を K とするとき，提供 QoS のスループット制約の定数は K の約数でなくてはならない．

（3） 要求 QoS に遅延制約が含まれるとき，対象システムの入出力間を結ぶ経路の中で，遅延関係がすべて与えられているような経路が少なくとも一つはなくてはいけない．

（4） 対象システムのコンポーネント接続関係について閉路はない．

制限 (1) をおく理由は，ジッタ制約からスループット制約を推論することはできるが，逆はできないためである．例えば，$\forall i \in \mathbb{N} : \ m \leq x_{i+1} - x_i \leq M$ から $\forall i \in \mathbb{N} : \ x_{i+10} - x_i \leq 10M$ は容易に推論できるが（発信間隔が M 以内ならば，10 回目の発信時刻は最初の発信時刻からは $10M$ 以内に起こる），逆は推論できない（10 回目の発信時刻が最初の発信時刻からは $10M$ 以内に起こるからといって，発信間隔が M 以内であるとはいえない）．制限 (2) は主に提案する変換法を簡単にするためのクラス制約であり，問題のインスタンスによっては緩めることができる．制限 (3) も，本質的には制限 (1) と同様の理由により設けている．

簡単な場合として，例 1 のように，与えられる QoS が遅延のみ（コンポーネント間遅延とコンポーネント内遅延）とする．このときクラス制限の (3)，(4) より，システム全体の遅延は，接続関係にある信号に対する変数の組に対して等式制約を加え，添字と全称子のない有限個の線形制約式で表現することができる．同様にして要求遅延制約式を得ることができる（例 1）．

ジッタ制約はスループット制約における定数 C が 1 である特別な形とみなすことができる．以降ではスループットで説明する．一般に QoS 式にスループット制約が含まれている場合は，定数 C が提供 QoS，要求 QoS ともに共通な場合は x_{i+C} に対応する変数を新たにおき，それに対して，同様にシステム出力までへの遅延制約式集合を生成することにより，対応可能である（例 2）．

[例 1] **提供 QoS 群**

$\forall i : \quad D_0 \leq b_i - a_i \leq D_1$
$\forall i : \quad D_2 \leq c_i - b_i \leq D_3$
$\forall i : \quad D_4 \leq d_i - c_i \leq D_5$
$\forall i : \quad D_6 \leq g_i - d_i \leq D_7$
$\forall i : \quad D_8 \leq f_i - e_i \leq D_9$
$\forall i : \quad D_{10} \leq g_i - f_i \leq D_{11}$

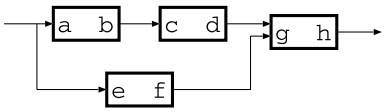

図 2　コンポーネント間遅延関係
Fig. 2　A delay configuration of components.

$\forall i:\quad D_{12} \le h_i - g_i \le D_{13}$

要求 QoS

$\forall i:\quad h_i - a_i < D_{14}$

図 2 のような遅延と上記の提供 QoS 群及び要求 QoS が与えられたとした場合，添字と全称限量子を無視することで線形制約式集合 P，Q を得る．

提供遅延制約式 P

$D_0 \le b - a \le D_1,\ D_2 \le c - b \le D_3$
$D_4 \le d - c \le D_5,\ D_6 \le g - d \le D_7$
$D_8 \le f - e \le D_9,\ D_{10} \le g - f \le D_{11}$
$D_{12} \le h - g \le D_{13}\ ,\qquad a = e$

要求遅延制約式 Q

$h - a < D_{14}$

提供 QoS と要求 QoS の整合性は $Q \supseteq P$ か否かにより判定する．P の補集合を考えれば，線形制約式の解空間の有無の問題となる．

［例 2］ 図 2 において提供スループット QoS と要求スループット QoS が次のように与えられたとした場合，システム出力までの遅延制約集合と提供スループット QoS から P′，Q′ を得る．

提供スループット QoS

$\forall i:\quad J_0 \le a_{i+6} - a_i \le J_1$

要求スループット QoS

$\forall i:\quad J_2 \le h_{i+6} - h_i \le J_3$

提供遅延制約式 P′

$J_0 \le a' - a \le J_1$
$D_0 \le b' - a' \le D_1,\ D_2 \le c' - b' \le D_3$
$D_4 \le d' - c' \le D_5,\ D_6 \le g' - d' \le D_7$
$D_8 \le f' - e' \le D_9,\ D_{10} \le g' - f' \le D_{11}$
$D_{12} \le h' - g' \le D_{13}\ ,\qquad a' = e'$

要求遅延制約式 Q′

$J_2 \le h' - h \le J_3$

例 1 と同様に P′，P の解集合を Q′ が含むか否かにより整合性を判定する．

スループット制約の定数 C（例 2 においては 6）が提供 QoS，要求 QoS で異なる場合，クラス制限の (2) より次のように提供 QoS 側の制約式集合を得る．

提供スループット制約式が $v_1 \le x_{i+k} - x_i \le v_2$ とする．また，k は要求スループット制約式の定数 n の約数とすると，提供 QoS 側の制約式集合として

$v_1 \le x_{j+k} - x_j \le v_2$
$v_1 \le x_{j+2k} - x_{j+k} \le v_2$
$v_1 \le x_{j+3k} - x_{j+2k} \le v_2$
$\cdots$
$v_1 \le x_{j+mk} - x_{j+(m-1)k} \le v_2\ (mk = n)$

を生成する．

4. 内部設計検証

4.1 UML ステートチャートによる各コンポーネントの動作仕様記述

3. において OCL を用いて定義された各コンポーネントの提供 QoS を満たすような動作振舞いを行う UML ステートチャートを記述する．本手法では，UML ステートチャートにクロックの概念を追加し，遷移に関してクロックの制約を付加できるように拡張する．

また，次節における検証では，ステートチャートを時間オートマトンに変換するため，遷移ラベルに記述する内容を次の形式に限定する．遷移ラベルに記述できないイベントトリガ，ガード条件，アクションに関しては，各状態のアクティビティ区画に記述する．

遷移ラベル：= イベントトリガ “[” ガード条件 “]” “/” アクション；
イベントトリガ：= 受信データ名；
ガード条件：= クロック変数に関する条件式；
アクション：= 送信データ名 | クロック変数の値書き換え；

受信データ名，送信データ名は最終的に UPPAAL のチャネルに置き換わる．

4.2 各コンポーネントの Timeliness QoS 整合性検証

本節での検証では，**3.** において OCL を用いて定義された各コンポーネントの timeliness QoS を **4.1** で記述された UML ステートチャートが満たしているかを確認する．本手法では，テストオートマトンの考えに基づいて時間オートマトンレベルで検証を行う．検証ツールには UPPAAL [2] を用いる．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 [9] が，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，timeliness 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 [10] 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．

また，テスト時間オートマトンに基づいた検証を行うため，OCL で記述された timeliness QoS から数値及びデータ名を抜粋し，本手法で扱う 3 カテゴリー (スループット，ジッタ，遅延) のためのテストオートマトンを生成する必要がある．本手法では，文献 [7] のテストオートマトンを採用する．スループット，ジッタをテストするためのテストオートマトンは図 3，遅延については図 4 である．

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 [7]．データ *in* がある一定区間 $RATE$ ごと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分かる．なお，文献 [8] 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．

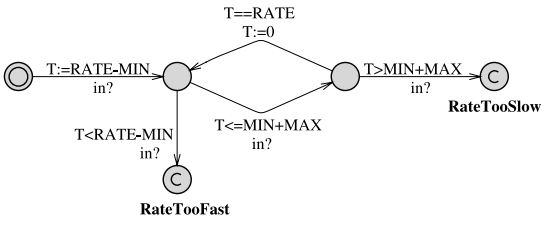

図 3　ジッタ，スループットのテスト時間オートマトン
Fig. 3　Obligation anchored jitter and throughput.

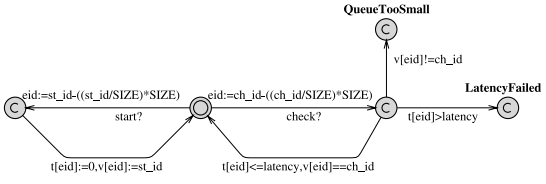

図 4　遅延のテスト時間オートマトン
Fig. 4　An automaton template for latency obligation.

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らず，図 4 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分かる．

4.3　検 証 手 法

各コンポーネントの接続関係を考慮したアプリケーション全体の検証は **3.3** で行っているため，各コンポーネントごとに単体テストを行えばよい．

検証方法は，テストオートマトンの原理に基づいて，次のようになる．まず各コンポーネントの動作仕様であるステートチャートを時間オートマトンへ等価変換する．これを単体で動かしデッドロックフリーであることを調べる．次に各コンポーネントに与えられている timeliness 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に対応する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 timeliness QoS を満たしていないことが原因である．

5.　時間制御コード生成系

4.2 での検証により各コンポーネントに対して与えられた timeliness QoS を満たしていればアプリケーションで要求されている timeliness QoS が満たされていることが保証できるので，プログラムコード生成時

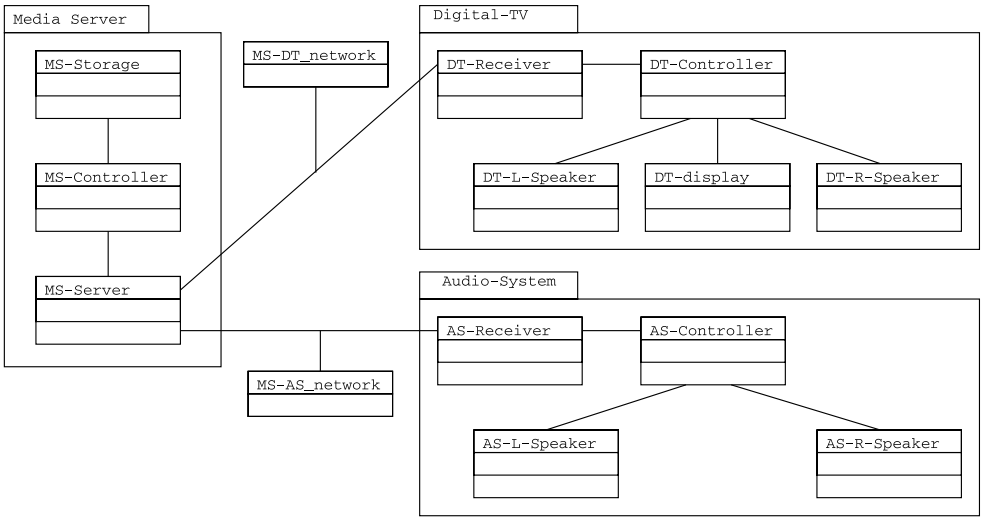

図 5　メディアサーバアプリケーションのクラス図
Fig. 5　A UML class diagram of media server application.

にこの方針を適用することが可能である．検証により，整合性が確認された timeliness QoS を満たすような時間制御が行われる Java プログラムを生成する [14]．分散実時間アプリケーションの生成を考慮し，各コンポーネントに対して記述された時間オートマトンの時間制御（クロック監視）は集中管理ではなく，各コンポーネントで自律的に行わせる．生成される制御プログラムでは，時間制約を満たせない場合は例外を発生させるようにしている．このような例外に対応して時間調整を行うような機能は設けていない．

6. 例題適用

実装した検証系，変換系及びコード生成系に対して例題を適用した．なお，実験環境は OS が Microsoft Windows XP Professional，CPU が PentiumIII 600 MHz 及びメモリが 384 MByte 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，ディジタルテレビ（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 [11]．各出力機器には指定された timeliness 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 5 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・ディジタルテレビ間及びメディアサーバ・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 外部設計検証

例題の timeliness 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–ディジタルテレビ間のネットワーク遅延は 100 ms 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 150 ms 以下である．

各々の timeliness QoS に関係する変数をクラス図に書き足し，それに対する制約として OCL で記述すると次のようになる．

```
context MS-Server
  inv: self.throughput(MS-Serve_out, 1000) >=
  100
context MS-Storage
  inv: self.delay(MS-Storage_in, MS-Storage_out)
  <= 5
context MS-Server_DT-Receiver
  inv: network_delay <= 100
```

context MS-Server_AS=Receiver
inv: network_delay <= 150

これら OCL で記述された timeliness QoS の制約と UML クラス図で記述されたコンポーネント接続関係，更にアプリケーションに要求されている timeliness QoS を入力とし，アプリケーション全体の timeliness QoS の整合性に関して，線形制約式による検証を行う．例えば，要求されている timeliness QoS として，“ディジタルテレビのディスプレイ（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．この際，入力記述から検証式を生成するツールを作成し，線形計画問題の解法パッケージを用いた．結果は以下のとおりである．

検証時間　　：76 ms
線形制約式数　：115 個
利用変数　　：29 個

6.3 内部設計検証

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 timeliness QoS を満たすように記述する必要がある．例えば，図 6 はコンポーネント MS-Storage の動作仕様である．各コンポーネントに関する timeliness 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 timeliness QoS をテストオートマトンへ自動変換する．図 7 は，図 6 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　　：1153 ms
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，すべ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 で用いた検証系は，線形制約式の非可解性判定問題に帰着して検証するため，検証時間は生成される線形制約式の数に依存する．そこで検証アルゴリズムでは効率化を行い，生成される線形制約式の数を抑えている．そのため，入力となる timeliness QoS の数が変わらなければ，コンポーネントの数や接続関係の複雑さが異なっている場合でも検証に要する時間はほとんど変わることはない．

6.3 では，テストオートマトンを用いた検証を行っている．既存の手法では，アプリケーションを構成するすべてのコンポーネントとそれらに対するテストオートマトンを一度に並列動作させる [7] ため，状態爆発問題が起こる．実際，本論文で用いた例題に対して既存の手法で検証を行ったところ，状態爆発問題が起こり，検証を完了することはできなかった．一方，提案手法のように各コンポーネントごとにテストオートマトンで検証を行うと，状態爆発問題が起こることはなく数秒で完了した．なお，ステートチャートから時間オートマトンへの変換，及び時間オートマトンからのプログラムコード生成に対する処理時間は数秒であるがこれらの大部分は，入力となる XMI あるいは XML ファイルの解析に費やされている．

本手法における既手法との違いは，アプリケーショ

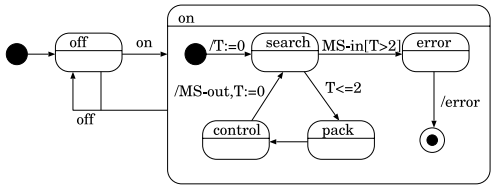

図 6　コンポーネント MS-Storage のステートチャート
Fig. 6　A UML statechart diagram of component MS-storage.

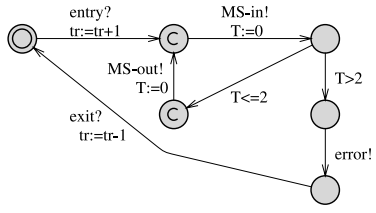

図 7　コンポーネント MS-Storage の UPPAAL 時間オートマトン
Fig. 7　An UPPAAL timed automaton of component MS-storage.

ン全体が満たすべき QoS 制約をコンポーネントレベルの QoS に分割し，それらを個々に検証することにより，状態爆発の可能性を著しく低下させている点である．QoS 制約の分割に際しては，接続関係が閉路に限られるなどの制限があるものの，本例題のようなシステムの検証としては有用であると考えられる．

また本手法では一般によく知られている UML，OCL 記述を用いて設計することができるので，ユーザは検証用の中間モデルであり，あまり知られていない拡張時間オートマトンの知識を必要としない．

以上，例題適用の結果，提案手法は検証時間の観点から見て有用であるといえる．

7. むすび

本論文では，要求される timeliness QoS を保証するための分散実時間アプリケーション開発手法を提案した．提案手法では，UML/OCL を用いた仕様記述，timeliness QoS 整合性検証，仕様どおりに制御を行う Java プログラムコード自動生成が順に行われる．この開発プロセスを踏むことにより，コンポーネント数が多く，それらの接続関係が複雑な場合においても効率的に開発することが可能となると思われる．

今後の課題として，現手法のような真偽判定だけでなく，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や確率時間オートマトンの解析技術 [12] の応用が考えられる．

謝辞　本学院生として，この研究の初期に従事した森一夫氏（現シャープ（株））に感謝致します．

文　献

[1] R. Alur and D.L. Dill, “A theory of timed automata,” Theoretical Computer Science, vol.125, pp.183–235, 1994.

[2] J. Bengtsson, K. Larsen, F. Larsson, P. Petersson, and W.Yi, “Uppaal – a tool suite for automatic verification of real-time systems,” LNCS 1066, pp.232–243, 1996.

[3] A.T. Campbell, G. Coulson, and D. Hutchison, “A quality of service architecture,” ACM SIGCOMM Computer Communications Review, vol.24, no.2, pp.6–27, 1994.

[4] A.W. Brown and K.C. Wallnau, “The current state of component-based software engineering,” IEEE Softw., vol.15, no.5, pp.37–46, 1998.

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

[6] Object Management Group, UML 2.0 Object Constraint Language (OCL), available at http://www.omg.org/

[7] D. Akehurst, J. Derrick, and A.G. Waters, “Design and verification of distributed multi-media systems,” LNCS, vol.2884, pp.276–292, 2003.

[8] B. Bordbar and K. Okano, “Verification of timeliness QoS properties in multimedia systems,” LNCS, vol.2885, pp.523–540, 2003.

[9] A. Wasowski, “On efficient program synthesis from statecharts,” Proc. 2003 ACM SIGPLAN Conf. on Language, Compiler, and Tool for Embedded Systems, vol.38, no.7, pp.163–170, 2003.

[10] 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.

[11] K. Havelund, A. Skou, K.G. Larsen, and K. Lund, “Formal modeling and analysis of an audio/video protocol: An industrial case study using UPPAAL,” BRICS Technical Report Series, RS-97-31, 1997.

[12] M. Kwiatkowska, G. Norman, and J. Sprotston, “Symbolic model checking for probabilistic timed automata,” LNCS, vol.3253, pp.293–308, 2004.

[13] 岡野浩三，森　一夫，谷口健一，“線形制約式を用いた時間 QoS 一貫性検証法，” 京大数解研講究録，vol.1375, pp.151–157, 2004.

[14] 牧寺　彩，岡野浩三，谷口健一，“分散環境実時間アプリケーション開発支援のための Timeliness QoS 一貫性検証系および時間制御コード生成系の実装，” 信学技報，vol.104, no.243, pp.19–24, 2004.

[15] E.M. Clark, O. Gumberg, and D.A. Peled, Model Checking, The MIT Press, 1999.

（平成 17 年 7 月 15 日受付，11 月 2 日再受付）

長井　栄吾　（学生員）

平 16 阪大・基礎工・情報卒．現在，同大大学院博士前期課程在学中．分散実時間システムの設計検証に関する研究に興味をもつ．

牧寺　彩

平 15 阪大・基礎工・情報中退．平 17 同大大学院前期課程了．現在，野村総合研究所勤務．在学中は分散実時間システムの開発法に関する研究に従事．

岡野　浩三　（正員）

平 2 阪大・基礎工・情報卒．平 5 同大大学院博士後期課程中退．同年同大助手，平 14 ケント大客員研究員．平 15 バーミンガム大客員講師．現在，阪大情報科学研究科助教授．工博．フォーマルアプローチによるソフトウェア設計開発などの研究に従事．情報処理学会，IEEE_CS 各会員．

谷口　健一　（正員：フェロー）

昭 40 阪大・工・電子卒．昭 45 同大大学院博士課程了．同年同大・基礎工・助手，同大基礎工，情報科学研究科教授を経て，平 17 より同大名誉教授．工博．この間，計算理論，ソフトウェアやハードウェアの仕様記述・実現・検証の代数的手法及び支援システム，関数型言語の処理系，分散システムや通信プロトコルの設計・検証法などに関する研究に従事．