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

信学技報
TECHNICAL REPORT OF IEICE.
SS2004-64 (2005-03)

UML/OCLを用いた分散実時間アプリケーション開発手法の提案

牧寺 彩† 長井 栄吾†† 岡野 浩三† 谷口 健一†

† 大阪大学大学院情報科学研究科
†† 大阪大学基礎工学部情報科学科
〒 560–8531 大阪府豊中市待兼山 1–3
E-mail: †{makidera,okano,taniguchi}@ist.osaka-u.ac.jp, ††e-nagai@ics.es.osaka-u.ac.jp

あらまし 本稿では，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から時間オートマトンを用いたUML/OCLに対する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，検証作業は二段階に分けられ，これにより検証の効率化がばかられる．本稿では，提案手法に基づいた検証系および導出系の実装，さら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，分散環境，実時間アプリケーション，Timeliness QoS，線形不等式，時間オートマトン，Java，コード生成

A Method to Develop Distributed Real-time Applications based on UML/OCL

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

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

Abstract We propose a technique for developing real-time application in distributed environments based UML/OCL. The technique efficiently checks Timeliness QoS consistency of the system and generates codes of the application. Checking Timeliness QoS consistency is devided into two steps, which allows us to verify it efficiently good performance. In this papaer, we will explain systems implementating the techniques. Applying the method to a simple example finds that checking the consistency and code generation perform in a few seconds.

Key words UML, OCL, Distributed Environment, Real-time Application, Timeliness QoS, Linear Inequality, Checking, Timed Automata, Java, Code Generation

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) を保証することがきわめて重要である．特に，Timeliness QoSに対する要求は非常に高い[3]．そこで，本稿では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コンポーネントベースシステムであることが多い[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(Unified Modeling Language)[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)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[6],[7]．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る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(Computation Tree

Logic) 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．

分散実時間アプリケーション設計および開発は本提案手法を用いた以下の手順にしたがって容易に行えるようになる．

（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は UML クラス図を用いて与える．さらに，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 [12]．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る舞い仕様を，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 [8]．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および動作の自律性を実現し，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，アプリケーション全体として動作仕様記述どおりに時間制御を行う．

本稿では，提案手法にもとづいた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．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，時間の観点からみて有用であることが分かった．

以降，2. 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，3. 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および各コンポーネントの提供 QoS 整合性検証系，4. 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．5. において提案手法を例題に適用する．最後に，6. でまとめる．

2. 外部設計検証

2.1 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．

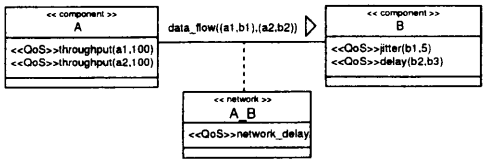

図 1 クラス図によるコンポーネント間接続関係

Fig. 1 A Configuration of Components in UML Class Diagram

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．コンポーネント間の接続関係は関連 (Association) をもって指定する．

2.2 コンポーネントおよびネットワークに対する Timeliness QoS の記述

次に，各コンポーネントおよ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) 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 Timeliness QoS を記述するという方針をとる．以下，記述方法について説明する．

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によって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 実数;
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コンポーネントおよび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．ただし，"--"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 100ms の間に送信する回数は 20 回以下
  inv: self.throughput(a2,100) ≦ 10
  --データ a2 を期間 100ms の間に送信する回数は 10 回以上
context B
  inv: self.jitter(b1, 5) < 1
  --5ms 間隔で受信するとき，ジッタは高々1ms
  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高々5ms
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高々100ms
```

2.3 線形計画法によるアプリケーション全体の Timeliness QoS 整合性検証

以下，本研究で扱うQoSカテゴリの線形制約式表現を示す．ただし，信号系列xについて，以下の性質を仮定する[1].

（1） 単調増加性　$\forall i \in \mathbb{N} : x_i < x_{i+1}$

（2） Non-Zenon性　$\forall K > 0 : \exists i : x_i > K$

（3） 非負値性　$\forall i \in \mathbb{N} : x_i \geqq 0$

スループット

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．

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

同様に，ある期間T内に信号$\mathbf{x}$が高々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．

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

ジッタ

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．

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

遅延

2つの信号$\mathbf{x}$，$\mathbf{y}$の遅延関係が高々Tであるという制約は次のように表現できる．

$$\forall i \in \mathbb{N} : 0 < x_i - y_{f(i)} \leqq T \ (f : i \text{についての関係式})$$

2.4 検証手法

ここでは，提供QoSと要求QoSの整合性を検証する方法を述べる．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下で要求QoSが満たされることを言う．

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[12]，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のクラスと接続関係の制約により線形不等式の数は有限に抑えられる．

3. 内部設計検証

3.1 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

2.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る舞いを行うUML Statechart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こととする．

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;

3.2 各コンポーネントの Timeliness QoS 整合性検証

本節での検証では，2.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3.1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もと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(図2).

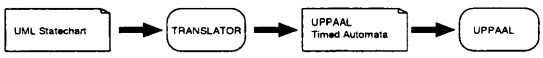

図2　検証系(UPPAAL)の入力への変換プロセス
Fig. 2　Translation Process from Statechart to UPPAAL

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，本手法では図 3 のように 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．

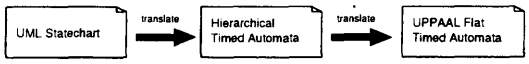

図 3　ステートチャートから UPPAAL 時間オートマトンへの変換プロセス
Fig. 3　Translation Process from Statechart to Time Automata

また，テスト時間オートマトンにもとづいた検証を行うため，OCL で記述された Timeliness QoS から数値およびデータ名を抜粋し，本手法で扱う 3 カテゴリ (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 [6] 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 4，遅延をテストするためのテストオートマトンは図 5 である．

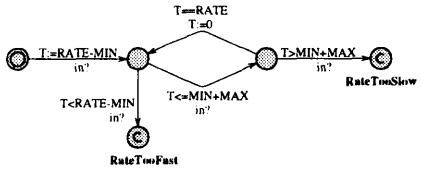

図 4　ジッタ，スループットのテスト時間オートマトン
Fig. 4　Obligation Anchored Jitter and Throughput

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 [6]．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 MIN を満たしているかを確認しながら状態遷移が行われる．RateToolow 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 [7] 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．

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 5 のテストオートマトンでは

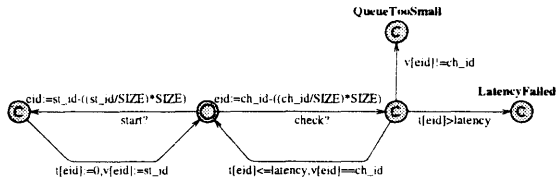

図 5　遅延のテスト時間オートマトン
Fig. 5　An Automaton Template for Latency Obligation

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．

3.3　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 2.4 で行っているため，各コンポーネント毎に単体テストを行えば良い．

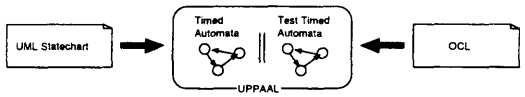

図 6　テストオートマトンを用いた UPPAAL での検証
Fig. 6　Verification on UPPAAL based on Test Automata

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．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．

4.　時間制御コード生成系

3.3 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 [12]．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．ただし，時間オートマトンネットワークにおいて同期の競合が発生する可能性もあるので，実行時に競合関係となる同期のみを集中管理するコンポーネントを新たに生成する．競合関係とならない同期は，各コンポーネントで処理される．

5. 例題適用

実装した検証系，変換系およびコード生成系に対して例題を適用した．なお，実験環境は以下の通りである．

OS : Microsoft Windows XP Professional
CPU : Pentium III 600MHz
Memory : 384MB

5.1 例題概略

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 [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 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．

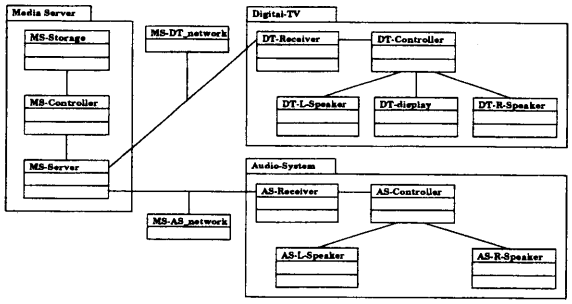

図 7 メディアサーバアプリケーションのクラス図
Fig. 7 A UML Class Diagram of Media Server Application

5.2 外部設計検証

例題における 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．

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，さら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．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

5.3 内部設計検証

5.2 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，5.2 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 8 はコンポーネント MS-Storage の動作仕様である．

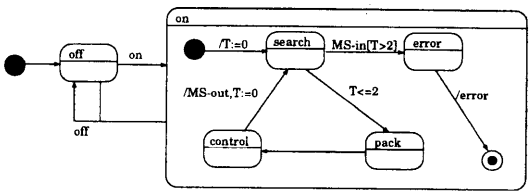

図 8 コンポーネント MS-Storage のステートチャート
Fig. 8 A UML Statechart Diagram of Component MS-Storage

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 9 および図 10 は，図 8 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

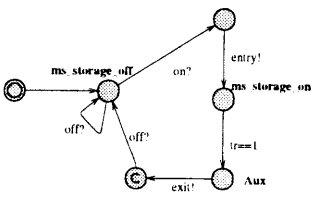

図 9 コンポーネント MS-Storage の UPPAAL 時間オートマトン (1)
Fig. 9 A Component MS-Storage of UPPAAL Timed Automaton (1)

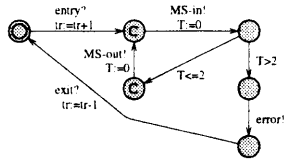

図 10　コンポーネント MS-Storage の UPPAAL 時間オートマトン (2)

Fig. 10　A Component of MS-Storage of UPPAAL Timed Automaton (2)

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．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．

5.4 考　　察

5.2 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．

5.3 における検証では，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全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 [6] 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，どのコンポーネントに対する検証も数秒で完了した．

また，ステートチャートから時間オートマトンへの変換，およ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．本例題では，各々90%の割合を占めることが判明した．この結果から，本質的な部分での計算時間はこの例題ではさほど大きくないということがわかった．

以上，例題適用の結果，提案手法は時間の観点からみて有用であることが期待できると言える．

6. おわりに

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，UML による仕様記述の拡張を行い，コンポーネント数が動的に変化するようなアプリケーションに対しても本手法を適用できるようにすること，また，現手法の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が考えられる．さらに，最終的には，本手法を搭載する分散実時間アプリケーション開発のための統合開発環境へ発展させたい．

文　　献

[1] R. Alur and D.L. Dill : "A Theory of Timed Automata," In Theoretical Computer Science 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] Andrew T. Campbell, Geoff Coulson, and David Hutchison: "A Quality of Service Architecture," ACM SIGCOMM Computer Communications Review, Vol.24(2), pp.6–27, 1994.

[4] Alan W. Brown and Kurt C. Wallnau: "The Current State of Component-Based Software Engineering," IEEE Software, Vol.15(5), pp.37–46, 1998.

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

[6] David Akehurst, John Derrick, and A. Gill Waters: "Design and Verification of Distributed Multi-media Systems," In Proceedings of the 5th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS '03), Lecture Note in Computer Science, Vol. 2884, pp.276–292, 2003.

[7] 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.

[8] Saehwa Kim, Sukjae Cho, and Seongsoo Hong: "Schedulability-Aware Mapping of Real-Time Object-Oriented Models to Multi-threaded Implementations," In Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications (RTCSA '00), pp.7–14, 2000.

[9] Andrzej Wasowski: "On Efficient Program Synthesis from Statecharts," In Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool for Embedded Systems (LCTES '03), Vol.38(7), pp.163–170, 2003.

[10] Alexandre David and M. Oliver Möller: "From HUPPAAL to UPPAAL: Translation from Hierarchical Timed Automata to Flat Timed Automata," BRICS Technical Report Series, RS-01-11, 2001.

[11] Klaus Havelund, Arne Skou, Kim G. Larsen, and Kristian 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] 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", 電子情報通信学会技術研究報告, Vol.104, No.243, pp.19–24, 2004.