

Title	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
Author(s)	牧寺, 彩
Citation	電子情報通信学会技術研究報告. SS, ソフトウェア サイエンス. 104(243) p19-p.24
Issue Date	2004-07-27
oaire:version	VoR
URL	<a href="https://hdl.handle.net/11094/27443">https://hdl.handle.net/11094/27443</a>
DOI	
rights	Copyright © 2004 IEICE
Note	

*Osaka University Knowledge Archive : OUKA*

<https://ir.library.osaka-u.ac.jp/>

Osaka University

## 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

牧寺 彩<sup>†</sup> 岡野 浩三<sup>†</sup> 谷口 健一<sup>†</sup>

<sup>†</sup> 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: †{makidera,okano,taniguchi}@ist.osaka-u.ac.jp

あらまし 著者らの研究グループでは、分散実時間アプリケーションにおける Timeliness QoS 一貫性を検証する手法、さらに、その保証された Timeliness QoS を満たす時間オートマトンから時間制約を満たしたプログラムコードを自動生成する手法を提案している。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。簡単なビデオ会議システムに適用した結果、それぞれ数秒で判定および導出が出来ることがわかった。

キーワード 分散環境, 実時間アプリケーション, Timeliness QoS, 線形不等式, 検証, 時間オートマトン, Java, コード生成

### A Timeliness QoS Consistency Checker and a Timing Control Code Generator for Development of Real-time Applications in Distributed Environments

Aya MAKIDERA<sup>†</sup>, Kozo OKANO<sup>†</sup>, and Ken-ichi TANIGUCHI<sup>†</sup>

<sup>†</sup> Graduate School of Information Science and Technology, Osaka University  
Machikane-yama 1-3, Toyonaka City, Osaka, 560-8531 Japan

E-mail: †{makidera,okano,taniguchi}@ist.osaka-u.ac.jp

**Abstract** Our research group has proposed a technique of checking Timeliness QoS consistency and that of generating codes from timed automata those have already been conformance checked for real-time applications in distributed environments. In this paper, we describe the implementations of them. Applying them to a simple video conference system example finds that both of them perform the tasks in a few seconds.

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

#### 1. はじめに

QoS(Quality of Service)の管理は分散実時間アプリケーションの設計において重要である。近年、アプリケーションを構成する各コンポーネントおよびアプリケーション全体に対し QoS の仕様記述方法として、UML Profile [1] や QML (QoS Modeling Language) [2] が提唱されている。一方、各コンポーネントの QoS 記述とアプリケーション全体で要求されている QoS の整合性や一貫性の検証に関する研究も必要であるが、現状ではまだまだ十分になされてい

ない。

また、実時間処理を扱う分散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動作系列の実行を指定した時間内に完了できる機構が必要となる。このため実時間アプリケーションの処理は煩雑なものとなり、何らかの手段で QoS に関する安全性が確認できたとしても、コーディング段階で安全性が保障されなくなることもある。手作業と比較し開発生産性や品質の向上が期待できるプログラムコード自動生成に関する研究がなされているものの、実時間アプリケーションを対象とした研究は少ない。

以上の背景により、研究グループでは、分散実時間アプリケーションにおける Timeliness QoS の一貫性を線形制約式を用いて検証する手法、さらに、その保証された Timeliness QoS を満たす時間オートマトン [3] から時間制約を満たした Java プログラムコードを自動生成する手法を提案している [10]。Timeliness QoS とは、QoS のうち時間に関するものを指す。

分散実時間アプリケーションの設計および開発は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) アプリケーションを構成する各コンポーネントで提供可能な Timeliness QoS(以降、提供 QoS)、およびアプリケーション全体で要求される Timeliness QoS(以降、要求 QoS)を時間変数を用いて線形制約式でそれぞれ記述し、提供 QoS が要求 QoS を満たしているか検証系で検証する。

(2) 検証により保証された Timeliness QoS をもとに、開発者は時間制約や動作を含む仕様を UPPAAL [4] 等のツールで時間オートマトンを記述する。

(3) (2) で記述した時間オートマトンを入力とし、コード生成系によって時間制御コードを自動生成する。ただし、各コンポーネントの動作については開発者がコード自動生成後に追加する。

提案手法により生成されるコードは、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ており、実行プログラムが異なる複数のクロックからなる環境、つまり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仕様記述どおりに時間制御が行われる。

本稿では、提案手法をもとに実装した検証系およびコード生成系について述べ、それぞれの例題適用を示す。結果、数秒以内に処理を終えることが確認でき、実用性が期待できると考える。

以降、2. では Timeliness QoS 一貫性検証系、3. では時間制御コード生成系について述べる。さらに、4. にて検証系、コード生成系それぞれの例題への適用を示し、5. でまとめる。

## 2. Timeliness QoS 一貫性検証系

### 2.1 Timeliness QoS

ネットワーク帯域、通信遅延、パケット損失率など定量的に表現できる通信品質のことを QoS(Quality of Service) という。これらはマルチメディアシステム等の性能に関係し、設計段階から考慮することが重要である。QoS のうち、時間に関するものを Timeliness QoS という。本研究では Timeliness QoS の中でもスループット、ジッタ、遅延の 3 カテゴリーのみを扱う。

図 1 はアプリケーションを構成するコンポーネントの接続関係、つまり入力信号や出力信号の受け渡し関係を表している。例えば図 1 中の“component 3”は、“component 1”から 1 個の入力信号を受け、“component 4”に 2 つの信号を渡す。

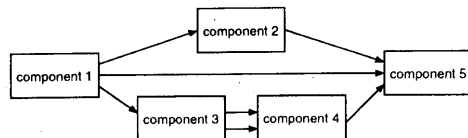


図 1 コンポーネントの接続関係

Fig. 1 A Configuration of Components

一般に、1 コンポーネントにはコンポーネントによって定まる  $n$  個の入力および  $m$  個の出力信号  $x^k (1 \leq k \leq n+m)$  が接続されているとする。なお、信号はこのように上付き添え字で区別するか、あるいは、誤解がなければ、 $x, y, z$  等の記号を用いる。また、各信号  $x^k$  に対して、変数  $x_i^k (i > 0)$  を導入する。これは信号  $x^k$  の  $i$  番目の発生時刻(例えば、フレーム出力信号における  $i$  番目のフレームの出力時刻)を表す。曖昧さを招かない限り、添え字  $k$  を省略し  $x_i$  と表す。なお、 $x^k$  は信号  $x^k$  の発生時刻系列を意味することとする。

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 [3]。

- (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  $\forall i \in \mathbb{N} : x_i \geq 0$

### スループット

ある期間  $T$  内に信号  $x$  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。

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

同様に、ある期間  $T$  内に信号  $x$  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。

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

### ジッタ

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

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

### 遅延

2 つの信号  $x, y$  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。

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

## 2.2 検証手法

### 2.2.1 検証方針

ここでは、提供 QoS と要求 QoS の一貫性を検証する方法を述べる。一貫性とは、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。

まず、要求 QoS、提供 QoS、およびコンポーネントの接続関係が与えられたとして、これらの線形制約式から一貫性検証式を生成する。このとき次の検証式を用いる。

$$\left( \bigwedge \text{提供 QoS の線形制約式} \right) \wedge \neg \left( \bigwedge \text{要求 QoS の線形制約式} \right)$$

この式は直観的に、要求 QoS を満たさないような場合があるかどうかを意味する論理式である。このため、一貫性を満たすかどうかの真偽と検証式の真偽は反転する。つまり、検証式が偽を返すときに限り一貫性が保証される。

ただし、検証系を実装するには各線形制約式に付いている全称子を除去する必要があるが、等価な線形制約式集合を生成すると無限個の式を生成することになる。しかし実際には、必要十分な有限個の線形制約式があれば検証することは可能であり、不必要な線形制約式を除外し検証を行えば良いことになる [10]。結局、検証式は線形制約式の非可解性判定問題に帰着する。

### 2.2.2 検証アルゴリズム

前節の検証方針ではいったん無限個の式を生成するが、現実的には不可能である。そこで、検証系で実装するアルゴリズムは、除外されることのない線形制約式のみ生成することで実現可能にする。

与えられた問題のインスタンスに対して、検証式に必要な線形制約式集合を生成する関数 `generateFormulae()` を次に示す。このアルゴリズムは、文献 [10] を改善したものである (詳細は文献を参照のこと)。

[入力] コンポーネント集合とコンポーネント間の接続関係が定義されたシステム  $S$ , 要求 QoS の線形制約式集合  $Re$ , 提供 QoS の線形制約式集合  $Pr$

[出力]  $Re$ ,  $Pr$  から生成された線形制約式集合  $lRe$ ,  $lPr$

```
function generateFormulae( $S, Re, Pr$ )
   $lRe := generateRequiredFormulae(Re)$ ;
   $T := buildTable(S, Re, Pr)$ ;
   $lPr := generateProvidedFormulae(S, T, Pr)$ ;
  return ( $lRe, lPr$ );
```

関数 `generateRequiredFormulae()`, および `generateProvidedFormulae()` を以下に示す。また、`buildTable()` は線形制約式に関する情報をテーブルに保持するための関数である。

```
function generateRequiredFormulae( $Re$ )
   $lRe := \emptyset$ 
  foreach  $c \in Re$ 
     $lRe := lRe \cup (\text{linear expression of } c)$ ;
  return  $lRe$ ;

function generateProvidedFormulae( $S, T, Pr$ )
   $lPr = \emptyset$ 
  foreach  $v$  s.t. end side component of  $S$ 
     $lPr := lPr \cup generateCompFormulae(v, T, \emptyset)$ ;
  return  $lPr$ ;
```

例えば、あるコンポーネントの出力  $x$  と入力  $y$  の間に次のような遅延がある場合、このコンポーネントの出力以降で現れる添字  $i$  に対し、入力以前の添字  $i$  ではなく  $i+2$  が関係する。

$$0 \leq x_i - y_{i+2} \leq 10$$

式生成時にはこのずれを考慮する必要がある。それゆえ `generateFormulae()` ではアプリケーションの出力あるいは入力で使用される添字と各コンポーネントの線形制約式で使用される信号変数の添字とのずれをテーブルに保持している。このテーブル作成をあらかじめ `buildTable()` で行い、その結果を受けてコンポーネントレベルの線形制約式を生成する (関数 `generateCompFormulae()`)。

```
function generateCompFormulae( $v, T, r$ )
  if  $v$  is start end of component then
    foreach Provided QoS  $c$  of  $v$ 
       $r := r \cup inequalities(c, T)$ ;
    else if there are no visited component
      return  $r$ ;
    else generateCompFormulae(Previous( $v$ ),  $T, r$ );
```

`generateCompFormulae()` は、引数に与えられたコンポーネントからアプリケーションの入力インターフェースを持つコンポーネントまでの全てのコンポーネントについて線形制約式を生成する再帰関数である。この関数では、まず処理するコンポーネントの出力インターフェース先のコンポーネントが全て処理済であるかを検査し、処理済ならば自コンポーネントの式生成を行う。このコンポーネントが複数の出力インターフェースを持つ場合、各インターフェース先のコンポーネントから複数呼び出されるため、最後のインターフェース先から呼び出されたときのみ式生成処理を行う仕組みとなる。関数 `inequalities()` は引数の  $c$  に対応する線形制約集合を引数の表  $T$  を参考にして生成する。`generateCompFormulae()` は自コンポーネント全ての制約式を生成した後で、入力インターフェース先の全てのコンポーネントについて `generateCompFormulae()` を再帰呼び出しする。

### 2.3 検証系の実装

これまでに述べた検証アルゴリズムにもとづいて検証系を実装した。この検証系はシステムの要求 QoS, 提供 QoS の線形制約式集合、およびコンポーネントの接続関係を入力とし、検証結果 (非可解性判定による真偽) を出力する。

検証系はまず、ユーザがテキスト形式で入力した各情報をデータベースに登録する。次に、データベースに登録された線形制約式集合からアルゴリズム `generateFormulae()` に基づいて検証式に必要な線形制約式集合を生成する。最後に、生成した線形制約式集合の非可解性判定を行う。非可解性判定には数理計画ソルバ LINDO [5] を用いた。

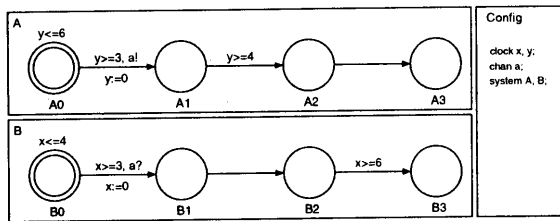


図2 時間オートマトンネットワーク

Fig.2 An Example of Timed Automaton Network

### 3. 時間制御コード生成系

#### 3.1 時間オートマトン

本研究では、コード生成系の入力となるコンポーネントの時間制御および動作振る舞いを拡張時間オートマトンネットワークで記述する。

時間オートマトンは実時間アプリケーションの動作仕様を形式的に記述できるモデルの一つである [3]。時間オートマトンによる動作モデルは、有限状態オートマトンに任意個のクロックとクロック制約を付加したもので、実時間アプリケーションの動的振る舞いを考慮した厳密な解析が可能である。また、拡張時間オートマトンネットワークにより、整数変数などの利用やコンポーネント間の同期の記述が可能となる。

図2はUPPAAL [4] による拡張時間オートマトンネットワークの記述例である。この例では、2つのコンポーネントA, Bから構成され、2つのクロックx, y, および一つの通信チャンネルaが用意されるようなアプリケーションが記述されている。クロックは動作開始時は0で初期化されており、単位時間ごとに1増加する。

UPPAALにおける遷移条件はguardと呼ばれ、例えば図2においてはクロックyの値が3以上である場合にのみ状態A0から状態A1への遷移が可能である。また、状態における時間制約はinvariantと呼ばれ、例えば状態A0ではyの値が6以下の間はA0に留まることができる。さらに、2コンポーネント間の同期は通信チャンネルにより実現可能である。図2では、遷移A0→A1, B0→B1において通信チャンネルaを介した同期が指定されている。

#### 3.2 コード生成手法

##### 3.2.1 生成方針

検証系によりTimeliness QoS一貫性の保証が確認されると、そのTimeliness QoSを満たすように時間制御が行われるJavaプログラム(以降、時間制御コード)を生成するフェーズに移る。コード生成に用いられる時間オートマトンネットワークは、保証されたTimeliness QoSとコンポーネントの接続関係にもとづいて開発者が記述するものとし、対応するコンポーネントの提供QoSを満たすか否かをモデルチェッカUPPAAL [4]を用いて検査を行っておくこととする。

各コンポーネントに対応する時間オートマトンは、それ

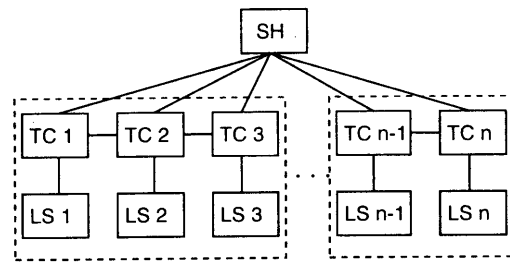


図3 生成コードのオブジェクト構成図

Fig.3 Structure of Generated Codes

ぞれにより表現される時間制約が提供QoSを満たしていさえすれば、動作振る舞いをどれだけ詳細に記述してもよい。ただし、コード生成後に動作振る舞いを開発者が追加することを考えると、オートマトンの1状態における動作振る舞いはJavaの1メソッドに対応することが望ましい。

##### 3.2.2 生成コードの実行方式

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。検証系により各コンポーネントが提供QoSを満たしていればアプリケーション全体の要求QoSが満たされていることが保証できるので、コード生成時にこの方針を適用することが可能である。ただし、時間オートマトン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。

図3は、コンポーネント数n個の場合の生成コード全体のオブジェクト構成図である。コード生成系は時間オートマトンの状態集合クラス(LS)、時間制御クラス(TC)のオブジェクト対をコンポーネント数だけ生成し、さらに同期処理クラス(SH)をただ一つ生成する。直観的には、状態集合クラスと時間制御クラスのオブジェクト対が1コンポーネントに対応する時間オートマトンに相当し、同期処理クラスオブジェクトはアプリケーション全体の同期競合調整コンポーネントとなる。点線で囲まれたオブジェクトは同一計算機上に配置されている。コンポーネント間通信は同一計算機同士ならばプロセス間通信、異なる計算機同士ならばネットワーク通信となる。以降、各クラスの動作について説明する。

##### 状態集合クラス(LS)

時間オートマトンの状態集合と各々に対応する処理が含まれる。ただし各状態に対応する処理は、コード生成後に開発者が記述する。対応する時間オートマトンの現状態を変数として保持し、現状態に応じた処理を実行する。現状態の処理終了を示すモード変数が用意されており、処理が終了するとTCオブジェクトからのメッセージを待つ。

##### 時間制御クラス(TC)

時間オートマトンのクロックを監視し、条件式や不変式

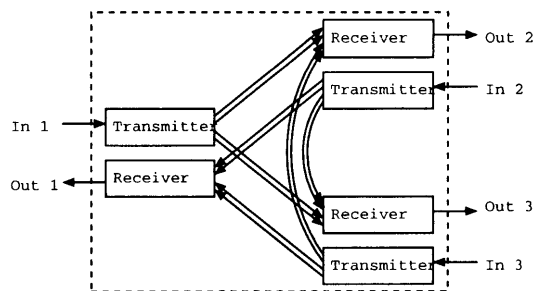


図4 3人用ビデオ会議システム

Fig. 4 A Configuration of the Video Conference System for three persons

をもとに状態遷移のスケジューリングを行う。LS オブジェクトから現状態と対応する処理が終了しているかどうかメッセージを受け取り、自身が監視しているクロックをもとに状態遷移可能か否か判定する。遷移可能ならばLS オブジェクトに状態遷移メッセージを渡す。また、クロック制約からデッドロックが発生した場合には例外を発生させる。例外発生後の処理については開発者が記述する。

競合の可能性がある同期については同期発生時に SH オブジェクトにメッセージを送信し、SH オブジェクトから同期実行許可のメッセージを受信した場合のみ実行する。

#### 同期処理クラス (SH)

競合関係にある同期の発生を監視し、優先度に応じて同期の調停を行う。TC オブジェクトから同期発生メッセージを受け取ると、同期実行許可の判定を行い、その結果をTC オブジェクトに返す。

#### 3.3 コード生成系の実装

実時間アプリケーションの動作仕様記述となる時間オートマトンネットワークおよび各コンポーネントの計算機配置は、UPPAAL を用いて記述が行われる。UPPAAL はこれらの情報をXML ファイルに保存する。実装したコード生成系はXML ファイルを入力とし、前節で述べた実行方式にもとづいたJava プログラムを生成する。

コード生成系はまず、入力となるXML ファイルの解析をDOM [6] を用いて行う。ここで解析された情報は内部データとして整理され、解析終了後ただちにJava プログラムを生成する。生成コードには、分散アプリケーションに適したミドルウェアプラットフォームCORBA [7] を導入した。CORBA は、DCOM [8] やJava RMI [9] を代表する他のミドルウェアプラットフォーム技術と比べ環境や言語に依存しないオープンな仕様を持ち、複数の環境や言語の混在した分散アプリケーションを構築することができる。また、アプリケーションから利用可能なサービスが豊富に用意されており、高機能な分散アプリケーション開発が容易になるという特徴を持つ。

#### 4. 例題適用

実装した検証系およびコード生成系に対して例題を適用した。なお、実験環境は以下のとおりである。

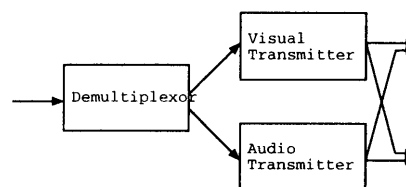


図5 送信部

Fig. 5 Transmitter

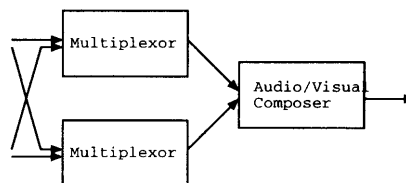


図6 受信部

Fig. 6 Receiver

OS : Microsoft Windows XP Professional

CPU : Pentium III 600MHz

Memory : 384MB

#### 4.1 例題概略

ビデオ会議システムにおいては、映像を画像データと音声データに分割する処理や、分割した各データを全ての受信端末あてに送信するといった処理を繰り返し行う。また、会議を行う人数によりシステムを構成するコンポーネント数が増加し、コンポーネント間の接続関係も複雑になることが考えられる。そこで、本稿ではビデオ会議システムの映像配信部を例題として取り上げ、評価を行う。

ビデオ会議システムのコンポーネント接続関係を図4に示す。図中のTransmitter およびReceiver は図5, 6のようなサブコンポーネントである。

本例題アプリケーションは、ある端末からシステムへ入力された動画を他の端末に配信するアプリケーションである。例えば、In1 からシステムへ入力された動画はOut2, Out3 に出力される。送信部コンポーネントでは、まず送信端末から入力された動画を画像データと音声データに分割する。各データは画像送信部、音声送信部から受信端末の受信部へと送信される。受信部コンポーネントでは、受け取った画像データと音声データの対を1動画として合成する。

#### 4.2 Timeliness QoS 一貫性検証

前節のようなシステムに対し、各コンポーネントの出力スループット、処理遅延、通信遅延を提供QoSとして記述する。さらに、要求QoSとして次をあげる。

「In2, In3 からシステムへ入力される動画フレームのスループットが25 フレーム毎秒以上であるとき、Out1 でシステムから出力される動画フレームのスループットは20 フレーム毎秒以上である。」

この例題と要求QoSに対し、検証系を適用した結果は以下のとおりである。

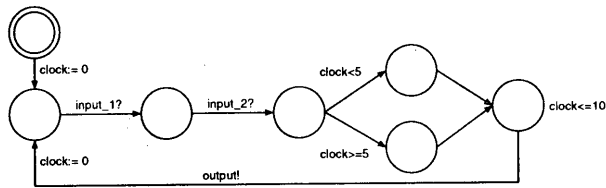


図7 Audio/Visual Composer の時間オートマトン記述

Fig.7 A Timed Automata of Audio/Visual Composer

検証時間 : 80 ミリ秒

生成した線形制約式 : 131 個

利用変数 : 50 個

検証系により提供 QoS と要求 QoS の一貫性が保証された。

#### 4.3 時間制御コード生成

保証された QoS のもとで、時間制御コードを生成する。各コンポーネントに対応する時間オートマトンを UPPAAL を用いて記述する。例えば、図7は受信部のサブコンポーネント Audio/Visual Composer の時間オートマトン記述である。このサブコンポーネントは2つの入力(画像および音声データ)を受け取ったあと2つのデータを合成し、10秒以内に出力を行う。全ての時間オートマトン記述を終えると、各コンポーネントの提供 QoS を満たしているか UPPAAL で検査しておく。

この例題をコード生成系に適用すると、以下の結果を得た。図8は、生成コードのオブジェクト構成図である。

コード生成時間 : 約1秒

コンポーネント数 : 31 個

時間オートマトン記述 : 5 個

時間オートマトン状態数 : 78 個

同期数 : 24 個

コード生成後、状態に対応する動作の記述を行い、分散環境に各コンポーネントを配置して動作を確認した結果、仕様どおりに時間制御を行う Java プログラムコードを生成したことを確認した。

#### 4.4 考察

研究グループで提案されている検証手法は、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、記述される提供 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らない。

コード生成系の処理時間の大部分は、入力となる XML ファイルの解析に費やされている。本例題では、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。

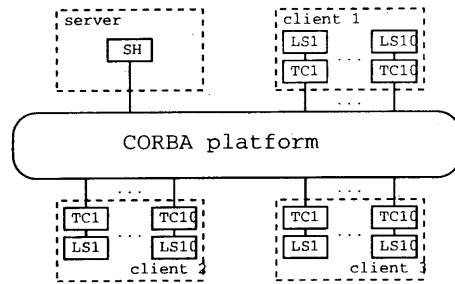


図8 例題適用結果

Fig.8 Structure of Generated Codes on CORBA

## 5. おわりに

研究グループでは、分散実時間アプリケーション開発のための Timeliness QoS 一貫性検証、および時間オートマトンネットワークによる動作記述からアプリケーションの時間制御を行うプログラムコードを生成する手法を提案している。本稿では、提案手法に基づいた検証系、コード生成系の実装について述べ、例題に適用した。結果、検証系については、コンポーネント数が多く、それらの接続関係が複雑なアプリケーションに対しても容易に提供 QoS と要求 QoS の一貫性を調べる事が可能であり、コード生成系については実装における開発者の負担を軽減することが期待できる。

今後の課題の一つとして、検証によって一貫性が保証された提供 QoS から時間オートマトンネットワークのテンプレートを自動生成し、検証、設計、実装を一貫して行うことのできる分散実時間アプリケーションのための統合開発環境へ拡張することが挙げられる。

## 文 献

- [1] Object Management Group : UML Profile for QoS and Fault Tolerance, available at <http://www.omg.org/>
- [2] S. Frolund and J. Koistinen : "Quality of Service Specification in Distributed Object Systems Design," Proc. of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS), pp.1-18, 1998.
- [3] R. Alur and D.L. Dill : "A Theory of Timed Automata," In Theoretical Computer Science 125, pp.183-235, 1994.
- [4] 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.
- [5] LINDO Systems Inc. : LINDO Linear Programming Solver API 2.0, available at <http://www.lindo.com/>
- [6] W3C : Document Object Model, available at <http://www.w3.org/DOM/>
- [7] Object Management Group : Common Object Request Broker Architecture (CORBA/IIOP) version 3.0.2, available at <http://www.omg.org/>
- [8] Microsoft Corporation : Distributed Component Object Model (DCOM), available at <http://www.microsoft.com/>
- [9] Sun Microsystems : Java Remote Method Invocation (Java RMI), available at <http://java.sun.com/>
- [10] 森 一夫, 岡野 浩三, 谷口 健一 : "マルチメディアシステムにおける Timeliness QoS 一貫性検証と時間制御コード導出", 信学技報, Vol.103, No.583, pp.13-18, 2004.