UMLと時間オートマトンを用いたソフトリアルタイムシステムの設計解析手法

坂倉賢昭[†,☆] 山根智[†]

多くの場面で用いられているリアルタイムシステムの中でも，近年，デッドラインに厳密でないソフトリアルタイムシステムが重要視されてきている．そこで，本研究では，ソフトリアルタイムシステムのUMLでの表記法を示し，そこからタスク構造化を行い，時間オートマトンに変換し，システムの性能を形式的に解析する手法を提案する．

The Design Verification Technique of a Soft Real-time System Using UML and Timed Automata

Masaaki Sakakura[†,☆] and Satoshi Yamane[†]

Recently, in the real-time systems used in many scenes, soft real-time systems, which are not strict by the deadline, are also important in recent years. So, in this paper, we show the notations in UML of soft real-time systems, and we design task structures from UML, also they are transformed into timed automata. Finally we formally propose the techniques for analyzing the performance of systems.

1. 序論

リアルタイムシステムはタスクに対する処理完了タイミングの要求である，デッドラインに対して厳格であるかどうかによって，ソフトリアルタイムシステムとハードリアルタイムシステムとに分けることができる[1)]．ハードリアルタイムシステムは，航空管制システムやプラント制御システムなど，人命に関わるような高い信頼性が要求されるシステムに用いられており，多くの研究がなされている．一方，ソフトリアルタイムシステムは，近年重要視されてきたものであり，画像や音声通信などのマルチメディアデータ処理，または分散データベースなどに用いられているが，モデル化が複雑なために確立した手法がないのが現状である．

近年では，リアルタイムシステムは大規模化，複雑化が進んでおり，設計が難しく，高い品質を保証する必要があるので，形式的手法をリアルタイムシステムの設計に用いることに関心が集まっている．しかし，実際に形式的手法を使って設計することは，まだ一般的ではなく研究段階にあるといえる．そのため，どのようなシステム設計法を用いて，その設計法のどの部分に形式的手法を適用するかが重要となる．また，最近では，生産性の向上が図れるUMLなどを用いたオブジェクト指向方法論にも期待が高まっている．そこで，本研究は，UMLで記述されたシステムをタスク構造化し，それを拡張した時間オートマトンによって表現し，形式的に性能を解析することにより，ソフトリアルタイムシステムの設計解析手法を確立することを目的とする．なお，設計解析とは，設計段階における性能解析であり，性能解析とは，スケジューラによって実行されるタスクの価値を計算することである．

本研究と関連性のある既存研究を紹介する．

ハードリアルタイムシステムの解析においては以下のような既存研究がある．

(1) 1999年，Yiらは，拡張した時間オートマトン（タスク付き時間オートマトン）の到達可能性解析によってハードリアルタイムシステムのスケジューラビィティ解析を行った[2)]．

(2) 2004年，Yiらは(1)の手法をタスクの順序制約，リソース制約も扱えるように拡張した[3)]．

ソフトリアルタイムシステムの解析においては，主に，関数を用いる手法[4)]と，確率を使う手法[5)]の2種類が提案されている．

† 金沢大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Kanazawa University
☆ 現在，京セラ株式会社
Presently with Kyocera Corporation

(3) 1985年，Jensenらはソフトリアルタイムシステムの性質を関数を用いて表現（価値関数）し，統計的解析でスケジューリングポリシの性能評価を行った[4]．
(4) 1999年，Liuらが，タスクがデッドラインをある確率以上で満たすか，という形でソフトリアルタイムシステムを解析する手法を提案し，PERTS real-time prototyping環境で実装を行った[5]．
(5) 2005年，我々は，Yiらの手法[2]を価値関数で拡張して，ソフトリアルタイムシステムの性能解析手法を提案した[6]．

また，価値関数を用いたスケジューリングポリシとして次のものが提案されている．
(6) 1996年，Mossezらによって価値関数によるスケジューリングポリシ（Value-Density Algorithms）が提案されている[7]．

一方，オブジェクト指向開発手法をリアルタイムソフトウェア設計に適用する研究は過去，多数あり，歴史的には，以下の主要な流れがある．
(7) 1991年に，初期のリアルタイムシステムに特化したオブジェクト指向開発手法としてShlaer-Mellor法が提案されている[8]．これはリアルタイムシステムを静的な側面によりシステム分析する手法である．
(8) 1994年に，Selicらは Shlaer-Mellor法を発展させて，ROOMを提案している[9]．これはアクターオブジェクトとプロトコルオブジェクトを用いて，動的な振舞いを厳密にモデル化して，その実行メカニズムを提供している．
(9) 1996年に，Awadらが OCTOPUSを提案しており，分析からタスク設計までサポートしている，最初の手法である[10]．これは，オブジェクトの並行性に着目して，オブジェクトをグループ化してタスク設計を行っている．
(10) 1997年に，OMGは様々なオブジェクト指向開発手法を統合化して，UML1.1として標準化した[11]．
(11) 2000年に，Gomaa は OCTOPUSを発展させて，COMETを提案しており，分析からタスク設計までサポートしている[12]．これは，UMLを使ってオブジェクトを設計して，その後，オブジェクトベースでタスク構造化基準により，タスク設計法を提示している．
(12) 2003年に，OMGが UML/SPTを提案して，スケジューラビリティやパフォーマンス性をモデル化できる枠組みを提供している[13]．
(13) 2004年に，Douglassがリアルタイム UMLの中に UML/SPTを盛り込んだ手法を提案している[14]．この手法では，UMLのほとんどすべての図を使って，スレッドを構成して，オブジェクトに割り当てて，実行モデルを作成する．
(14) 2006年，Kimらは，COMETと異なり，シナリオ（オブジェクトの動的な流れ）をベースにしたタスク構造化手法，Scenario-based Implementation Synthesis Architecture（SISA）を提案した[15]．

本研究では，まず，OMGの UML/SPTで価値関数を表現して，ソフトリアルタイムシステムを記述し，Gomaaの COMETの考えを応用し，オブジェクトベースでタスク構造化を行う．これは，ソフトリアルタイムシステムのように，オブジェクト間の通信が頻繁であったり，排他制御や共有オブジェクトがあったりするシステムには，オブジェクトベースの考え方がシナリオベースよりも適していると考えられるからである．次に，タスク構造をモデルで表現する．モデルには，(2)の手法の時間オートマンを (3)の価値関数で拡張した (5)（以後，拡張時間オートマトン）を用い，スケジューリングポリシとしては，(6)の Value-Density Algorithmsを用いる．最後に，そのモデルで解析を行い，ソフトリアルタイムシステムの性能を評価する．ソフトリアルタイムシステムの性質を価値関数で表現したものを，UMLで記述，また，そこからオートマトンへの変換，記号的解析は行われたことがなく，本研究には新規性がある．

一方，UMLでの開発手法に時間オートマトンでの検証を含めたものに次のものがある．
(15) 2006年，大阪大学の岡野や谷口らによって，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時間制約への整合性検証，時間制御コード自動生成を行う手法が提案されている[16]．本研究は，ハードリアルタイムシステムの Timelinenss QoSの検証であり，我々のソフトリアルタイムシステムの性能研究とはまったく異なる．

以降での本稿の構成は以下である．2章ではソフトリアルタイムシステムの UMLを用いた設計手法を提案し，それをタスク構造化するまでの流れを説明する．3章では，ソフトリアルタイムシステムを表現するための拡張時間オートマトンと，その解析方法について説明する．4章では，簡単な例を基に本研究を実証を行う．最後に，5章ではまとめと今後の課題について述べる．

2. UMLからのタスク構成手法とその拡張時間オートマトン記述

本章では，UMLから拡張時間オートマトンへ変換

していくまでの流れを説明していく．具体的には，まず，UML により作成したソフトリアルタイムシステムのオブジェクト指向モデル（設計モデル）を作成する．次に，タスク構造化基準に従ってタスクを決定し，設計モデルからタスク構造図を作成する．そして，タスク構造図を拡張時間オートマトンへ変換していく．本章では，タスク構造化における価値関数の扱い方が本研究のオリジナルである．本章では 2.1 節で UML モデリングについて，2.2 節でタスク構造化について，2.3 節で拡張時間オートマトンへの変換方法について説明する．

2.1 UML

UML（Unified Modeling Language）とは，オブジェクト指向のシステム開発における，ソフトウェア設計図の統一表記法である．従来，オブジェクト指向設計の表記法は 50 以上が乱立していたが，1997 年 11 月に OMG（Object Management Group）によって UML が標準として認定された．それ以来，いくつかの改定と改良を経て，2006 年現在リリース 2.0 に至っている．本研究では UML2.0 を用いていく．UML2.0 では，6 種類の構造図と 7 種類の振舞い図が定義されており，それぞれ静的な側面を扱う図（クラス図，コンポーネント図，コンポジット構造図，配置図，パッケージ図，オブジェクト図）と動的な側面を扱う図（アクティビティ図，コミュニケーション図，相互作用概念図，シーケンス図，状態マシン図，タイミング図，ユースケース図）に分けられる[17]．

本稿では，以下のように UML を用いる．

(1) オブジェクトの静的な構造をクラス図，オブジェクト図で記述して，オブジェクトの動的な振舞いをシーケンス図で記述する．

(2) リアルタイムシステムは時間制約をともなった処理を行うシステムであることから，モデル上に時間制約を記述できる必要がある．この場合，ノート，タグ付き値，および制約といった UML の要素によって記述する．また，シーケンス図上では，メッセージ間の時間間隔を記述することで時間制約を表す．

2.1.1 静的構造図

クラスとは，共通の属性と操作を持った一群のオブジェクトを抽象的に定義したものである．そして，クラス図とはシステムを構成するクラス間の関連を表したものである．一方，オブジェクト図は，特定の時点におけるオブジェクト間のリンクを表したものである．これら 2 つの図をまとめて静的構造図と呼ぶ．静的構造図は，システムをオブジェクト指向で分析・設計するうえで主要な図となる．**図 1** にクラス図の例を，**図 2** にオブジェクト図の例を示す．

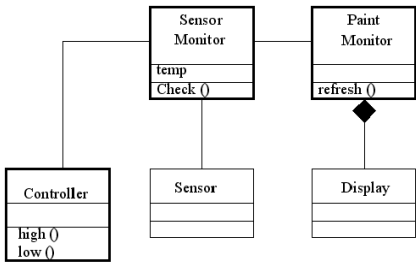

図 1 クラス図
Fig. 1 Class diagram.

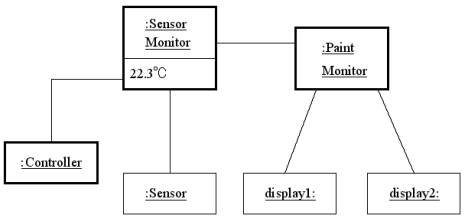

図 2 オブジェクト図
Fig. 2 Object diagram.

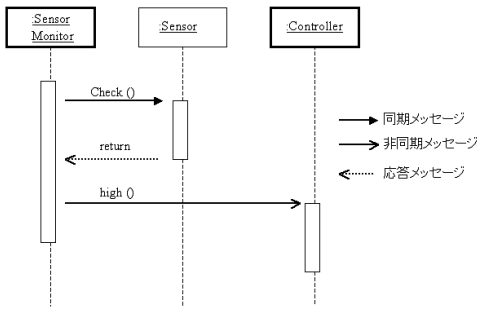

図 3 シーケンス図
Fig. 3 Sequence diagram.

クラスには，アクティブクラスとパッシブクラスの 2 つがある．同様にそのクラスから生成されるオブジェクトもアクティブオブジェクトとパッシブオブジェクトになる．アクティブオブジェクトとは制御アクティビティを開始できるオブジェクトのことである．アクティブクラスとアクティブオブジェクトは，クラス枠およびオブジェクト枠を太線で囲んで表記する．

2.1.2 シーケンス図

シーケンス図とは，オブジェクトの動的な振舞いを時系列で表したものである．シーケンス図では，上から下へと時間が流れ，オブジェクトの存在期間は生存線で表して，メッセージとそれらのシーケンス（連続して起こる順序）に重点が置かれている．そのことから，シナリオにおける，オブジェクト間のメッセージのやりとりが理解しやすい．**図 3** にシーケンス図の

例を示す．

メッセージとは，オブジェクトからオブジェクトへと送られるデータ及び制御情報の両方，あるいは，いずれか一方を抽象化したもので，以下の 3 つに分類される．

同期メッセージ

送り側がメッセージを送信した後，受け側が応答メッセージを返すまで待たされるメッセージ．

非同期メッセージ

送り側がメッセージを送信しても，受け側の応答メッセージを待たないメッセージ．

応答メッセージ

同期・非同期メッセージの受け側が，送り側へ応答を返すメッセージ．

本稿では，メッセージとして，関数呼び出し，リアルタイム OS を介したイベント（タスク制御），および割込みなどを考える．

2.1.3 UML 図での時間表現

本稿で使用するダイアグラム上の時間表現は UML/SPT を用いる[13]．ノート，タグ付き値，および制約といった UML の要素を使って時間制約や価値を記述する．以下に，それら 3 つの要素を説明する．

ノート

ノートは，意味的な影響を与えないダイアグラム上の要素であり，右上隅を折り曲げた四角形で表される．ノートを使うことで，ダイアグラムにテキストの注釈を付け，モデルを理解しやすくする．

タグ付き値

タグ付き値は，UML の要素にユーザが付け加えたプロパティを表現することができる．ノート内に { プロパティ＝値 } の形式で表現する．

制約

制約は，UML の要素に対して適用される何らかの付加的制限（通常の UML で決められている規則以上のもの）を表現することができる．つねに {} で囲んで表記されるが，ノート内に記載されることもある．

2.1.4 価 値 関 数

ソフトリアルタイムシステムの性質を表現するための価値関数について説明する．ソフトリアルタイムシステムはタスクの処理がデッドラインをミスしてもエラーにならないが，実行価値が下がってしまうものである．よって，それを表現するためにタスクごとに価値関数というものを導入する．価値関数というのはタスクの実行価値の時間的変化を関数で表したものである．価値関数の例をあげておく（**図 4**）[4]．

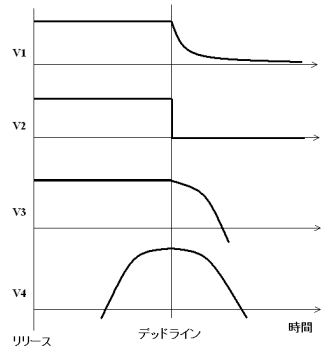

図 4 価値関数の例
Fig. 4 Example of utility functions.

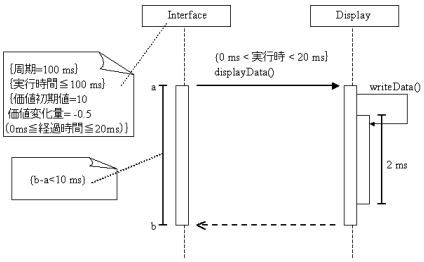

図 5 時間制約の記述例
Fig. 5 Example of specification of timing constraints.

- V1：カーナビの位置更新などのタスク．
- V2：センサの入力などのタスク．
- V3：ソフトリアルタイムのように価値が落ちていき負になるタスク．
- V4：衛星の発射などのタスク．

価値関数は本研究では，解析の簡単化のため，単調減少の一次関数のみを扱うこととする．また，価値関数の情報は UML 上ではノートに記述する．タグ付き値を用いて，経過時間 0 の値（価値初期値）と，範囲ごとの傾き（価値変化量）を示すようにする．今回は一次関数に限定したが，タグに記載法を変えれば単調減少であればすべての関数を扱うことができる．価値関数と時間制約の記述例を**図 5** に示す．

2.1.5 価値関数を用いたスケジューリングポリシ

ここで，価値関数を用いたスケジューリングポリシを説明しておく．価値関数を用いたスケジューリングポリシに以下の 2 つのものが提案されている[7]．

HVF（Highest Value First）

タスクの処理終了後の価値が待ち行列で最も高いタスクから実行していく．

HVDF（Highest Value Density First）

タスクの value density （価値/計算時間）を求め，待ち行列で最も高い value density をもつタ

スクから実行していく．

本稿では，価値を高めるのにより有効だと示されている[7]，HVDF をスケジューリングポリシとして用いる．

2.2 タスク構成手法

本節では，UML/SPT により作成した設計モデルから，Gomaa が考案したタスク構造化カテゴリ[12] に従って，新たに価値関数で拡張したタスクマッピングを行い，タスクを決定する．そして，そこからタスク構造図を作成していく．

2.2.1 タスクマッピング

リアルタイムシステムの設計では，並行性の観点からリアルタイム性を向上させるため，処理全体を機能ごとにタスクという理論的な実行資源に分割する．うまくタスク分割することで，優先順位に基づいた処理の切替えや，並行処理などが効率良く実現できる．そのため，論理的資源であるタスクをどう設計し構成するかを決定するタスク構造化が重要になる．しかし，現在の UML には，タスクの概念（分類子）がなく，UML モデル上でどう扱うかが決まっていない．そこで，Gomaa の考え[12] に従って，以下のタスク構造化手法を行う．

(1) システムにとって余計なタスクの存在は，コンテキストスイッチングを増やすことから，オーバヘッドが大きくなるため，効率が悪くなってしまう．そのため，設計者は，多数の単純なタスクとして設計するか，または，少数の複雑なタスクとして設計するかといったトレードオフに悩まされる．この場合，それらトレードオフに悩む設計者を助けるために考案された，タスク構造化カテゴリに基づくタスク構造化が望ましい．

(2) オブジェクト指向設計におけるタスク構造化では，タスクマッピングによって，オブジェクトをどのようにタスクに割り当てるか決定する．タスクマッピングのプロセスは，以下のような初期タスクマッピングと再タスクマッピングの 2 段階から成り立つとする．

  (a) 初期タスクマッピングでは，タスク候補となるアクティブオブジェクトを選出して，そのアクティブオブジェクトとタスクが 1 対 1 対応になるようマッピングを行う．

  (b) 再タスクマッピングでは，タスク候補を結合して，物理タスク（実際のタスク）の数を減らすためのマッピングを行う．

本研究では，次の Gomaa が考案したタスク構造化カテゴリ[12] に基づいて，本研究の要素として価値関数を考慮して，タスクマッピングを行う．

Definition2.2.1 （タスク構造化カテゴリ）[12]
タスク構造化カテゴリとは，UML モデル上でタスク候補となるオブジェクトの選出基準と，効率化を図るためのタスク候補の結合基準を定めたものであり，以下のような 5 つのタスク構造化基準に分類される．

(1) I/O Task Structuring Criteria
どのようにインタフェースオブジェクトが I/O タスクに割り当てられるか，そして，いつ I/O タスクが起動するかを扱う．

(2) Internal Task Structuring Criteria
どのように内部オブジェクトが内部タスクに割り当てられるかを扱う．

(3) Task Priority Criteria
タスクに対する優先度（実行重要度）を扱う．

(4) Task Clustering Criteria
どのようにオブジェクトを並行タスクにグループ化するべきかを扱う．

(5) Task Inversion Criteria
タスク初期設計，または，タスク再設計における，タスクのオーバヘッドを減らすためのタスクマージを扱う．

□

今回はタスクの優先度はスケジューリングポリシの HVDF に従って決めるので，(3) は採用しない．また，Gomaa の考え[12] によると，タスクマージは最適化のためであり，処理としては，本来では望ましくはないタスク結合である．よって，今回は採用しないものとする．したがって，初期タスクマッピングに対して，(1) I/O Task Structuring Criteria，(2) Internal Task Structuring Criteria を適用して，再タスクマッピングに対して，(4) Task Clustering Criteria を適用する．以降では，初期タスクマッピングおよび再タスクマッピングについて説明する．

2.2.2 初期タスクマッピング

初期タスクマッピングでは，タスク候補となるアクティブオブジェクトを選出して，そのアクティブオブジェクトとタスクが 1 対 1 対応となるマッピングを行う．まず，タスク候補となるアクティブオブジェクトを検討する．一般的には，Gomaa の提案[12] のタスク候補の種類を価値関数により修正した**表 1** に，1 つまたは複数が該当する処理が存在すれば，それがタスク候補となる．これを基準に，設計モデルからタスク候補となるアクティブオブジェクトを選出する．

表 1 タスク候補の種類と特徴[12)]
Table 1 Task candidate's kind and characteristic.

タスク候補の種類	特徴
Asynchronous I/O Device Interface Task	デバイスからの割込みによって起動
Periodic I/O Device Interface Task	タイマイベントによって起動
Passive I/O Device Interface Task	計算と入出力の処理を重複させる
Resource Monitor Task	リソースの入出力要求を調整する
Periodic Task	周期的なアクティビティを持つ
Asynchronous Task Contorol Task	要求駆動アクティビティを持つ 状態依存した処理を制御する
User Interface Task	シーケンシャルな アクティビティを持つ
Time-Critical Task	価値関数を持つ

タスク候補となるアクティブオブジェクトが決定されると，そのオブジェクトに関わるUMLの要素を計算時間や価値関数，周期に対応できる．

2.2.3 再タスクマッピング

初期タスクマッピングでは，多くのオブジェクトがタスク候補であるアクティブオブジェクトとして選出されることがある．この場合，多数の単純なタスクをともなうため，以下のような弊害が考えられる．

(1) 制御が複雑になる．
(2) タスク間でリソースの競合が発生しやすくなる．
(3) メモリを圧迫する．
(4) コンテキストスイッチング（タスク間の切替え）に要するオーバヘッドが増える．

以上のような弊害を抑えるため，Gomaaの提案[12)]の再タスクマッピングでは，タスク候補を結合して，物理タスクの数を減らすためのマッピングを行う．その際に，本稿オリジナルであるが，価値関数を合成する．具体的には，定義(4) Task Clustering Criteriaに基づいて物理タスクへグループ化を行う．物理タスクへのグループ化とは，複数のタスク候補のアクティブオブジェクトの代わりに，新たに1つのアクティブオブジェクト（物理タスク）を設けることである．今回，Task Clustering Criteriaとして以下のものを用いる．

(1) **Temporal Clustering**
  (a) 2つ以上のタスク候補が同じ周期によって起動される場合は，それらを1つの物理タスクへグループ化する．
  (b) 2つ以上のタスク候補が同じイベント（たとえば，タイマイベント）によって起動される場合は，それらを1つの物理タスクにグループ化する．

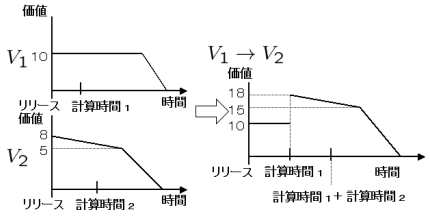

図 6 価値関数の加算の例
Fig. 6 Example of addition of utility functions.

(2) **Sequential Clustering**
システム要求から，いくつかのタスク候補のアクションにシーケンシャルな順番で実行する制約がある場合は，シーケンスの起点となるオブジェクトをアクティブオブジェクトとして，それらを1つの物理タスクへグループ化する．

(3) **Mutually Exclusive Clustering**
システム要求から，いくつかのタスク候補の間に，排他的に実行する制約がある場合は，それらを1つの物理タスクへグループ化する．

本研究では，タスクは価値関数を持っているので，グループ化の際の価値関数の扱いを考える必要がある．また，タスクは逐次処理で行うため，複数のタスクをまとめる場合，実行順序も決める必要がある．以下のように，グループ化の際の価値関数とタスクの実行順序を扱うことにする．

(1) 複数のタスクを実行することになるので，実行順序は経過時間0でのvalue densityが高いものから順に行うこととする．価値関数は，実行順に，実行時間分ずつずらして加算することにする（**図 6** 参照）．
(2) 実行順は決まっているので，(1)同様，価値関数を加算する．
(3) 排他的に実行するので，実行順序，価値関数に変化はない．ただし，前提条件として，排他的に実行するこれらの処理は同じ価値関数を持つとする（たとえば，データベースへの書き換え処理が排他的処理の場合，どのオブジェクトから処理が行われても同じ価値関数とする）．

タスクマッピングの例を**図 7** に示す．左は初期タスクマッピングを行ったオブジェクト図で，太線のオブジェクト2つがタスク候補にあたる．右は，左の図から，再タスクマッピングを行ったもので，1つの太線のオブジェクトが物理タスクである．

2.2.4 タスク構造図の生成

前述により，タスクマッピングにより，アクティブオブジェクトからタスクを決定した．次に，オブジェクト

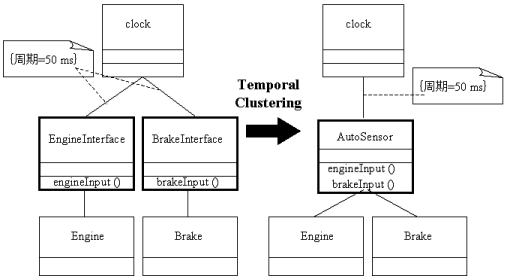

図 7 タスクマッピングの例
Fig. 7 Example of task mapping.

指向モデルのタスク構造を明確に示すために，Gomaa の提案[12] に従って，各種ダイアグラムからタスク構造図を生成する．本稿では，シーケンス図はプロトコル設計などにおいても幅広く使われており，多数の設計者になじみやすいので，シーケンス図からタスク構造図を生成する．なお，コミュニケーション図からもタスク構造図を生成できる．タスク構造図の構成法としては，まず，タスクと関連するオブジェクトは，オブジェクト図の静的構造を使って割り出す．次に，メッセージのメッセージタイプは，シーケンス図からメッセージをどのような通信の仕方で実装するかによって決定する．メッセージタイプとは，通信の仕方を表現するため，より具体的にメッセージを分類したもので，関数呼び出し，同期，非同期，リターンの 4 つのタイプがある．本稿では，以下のように各メッセージのメッセージタイプを決める．ただし，タスクから共通関数を呼び出す場合があるので，メッセージタイプの (1) を本稿のオリジナルとして追加する．なお，(2) から (4) は Gomaa の提案[12] である．

(1) メッセージタイプ f（関数呼び出し）
タスクとタスクでないオブジェクトとの間のメッセージ．

(2) メッセージタイプ s（同期）
タスク間の同期メッセージ．

(3) メッセージタイプ a（非同期）
タスク間の非同期メッセージ．

(4) メッセージタイプ r（リターン）
f か s の応答メッセージ．

タスク構造図上では，それぞれ，f，s，a，r をメッセージに付けて表記する．**図 8** にタスク構造図の例を示す．

次に，どのようにシーケンス図からタスク構造図を生成するかを定義する．

Definition2.2.2 （タスク構造図の生成法）
タスクとオブジェクトとの通信パターンは，以下の 4 通りなので，4 つに分類して，シーケンス図からタスク構造図を生成する方法を定義する．

(1) タスクとオブジェクト間，または，タスク間に通信がない場合，**図 9** のようにタスク構造図を作る．
(2) タスクとオブジェクト間の関数呼び出し通信の場合，**図 10** のようにタスク構造図を作る．
(3) タスク間の同期通信の場合，**図 11** のようにタスク構造図を作る．
(4) タスク間の非同期通信の場合，**図 12** のようにタスク構造図を作る．

□

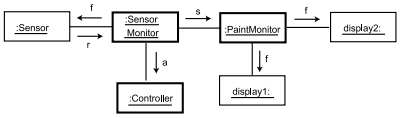

図 8 タスク構造図の例
Fig. 8 Example of task structure diagram.

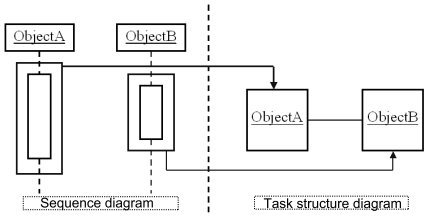

図 9 通信なしの生成
Fig. 9 Construction without communications.

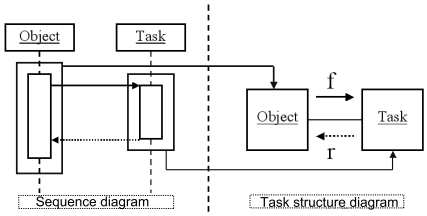

図 10 関数呼び出し通信の生成
Fig. 10 Construction of communications with function calls.

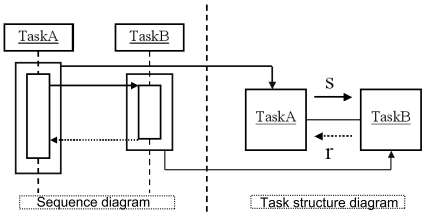

図 11 同期通信の生成
Fig. 11 Construction of synchronous communications.

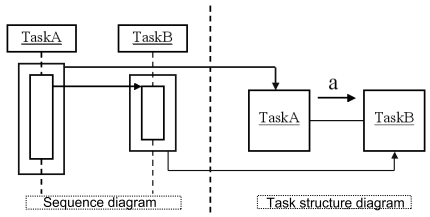

図 12 非同期通信の生成
Fig. 12 Construction of asynchronous communications.

2.2.5 拡張時間オートマトン構成手法

生成したタスク構造図より拡張時間オートマトンを構成する．ここでは，本研究でタスク構造図からどのように拡張時間オートマトンを構成するかを説明するだけにとどめる．拡張時間オートマトンやタスク仕様の形式的記述手法に関しては，次章で詳しく説明する．

本研究では以下の規則に従って，タスク構造図からタスク仕様の拡張時間オートマトンを構成する．

(1) 本研究でのタスク仕様の拡張時間オートマトンとは，システムのタスク（内部タスクと外部タスク）の到着（リリース）のタイミングとパターンを表現したオートマトンである．よって，原則として，タスク構造図の 1 つのタスクが拡張時間オートマトンの 1 つのノード（後述の Definition3.1.2 参照）に対応する．

(2) タスク間には同期通信と非同期通信の 2 種類がある．これらはメッセージをプールしておくバッファを用いて表現する．

(3) タスクから呼び出されるオブジェクトには，特定のタスクからのみアクセスされるもの，複数のタスクからアクセスされるものの 2 種類がある．

- 前者の場合では，タスクのノードにアクセスされるオブジェクトをそのタスクに含めるようにする．すなわち，そのノードがタスクとオブジェクトの処理両方に相当することになる．
- 後者の場合でも，同様にタスクのノードにアクセスされるオブジェクトをそのタスクに含めるようにする．しかし，当然ながら，タスク間で共有されるオブジェクトの場合はタスク間で競合が発生する場合がある．このため，タスクからのアクセスに対して，共有オブジェクトに排他制御をかける必要がある．本研究では，セマフォアを用いてスケジューラによって共有オブジェクトの排他制御を実現する．

3. 拡張時間オートマトンと性能解析手法

本章では，前章であげた拡張時間オートマトンについて，また，拡張時間オートマトン[6] の性能解析手法について説明していく．

まず，本解析手法の対象となるシステムの前提条件を述べる．本研究ではタスクは周期タスクと非周期タスクの両方を扱うが，非周期タスクはタスクが来てから，最短でもこの期間は次のタスクが来ることがないという最小周期が分かっているものとする．また，タスクが処理されずに，タスクキューが延々長くなっていくことがないとする．以降，本章は，3.1 節で拡張時間オートマトンの仕様について，3.2 節で解析について説明する．

3.1 拡張時間オートマトン

拡張時間オートマトンについて説明していく．本手法の拡張時間オートマトンと文献 2) のタスク付き時間オートマトンとの違いは，対象がハードリアルタイムシステムからソフトリアルタイムシステムに変わったため，タスクの要素として，デッドラインの代わりに，価値関数を持っている点である．

3.1.1 構　　文

まずタスクについて定義する．

Definition3.1.1 （タスク）

タスク $P_i \in P$ は $(c_{P^i}, V_i(t_i))$ の組で表す．

c_{P^i}　タスク P_i の残り計算時間

$V_i(t_i)$　タスク P_i の現在の価値（t_i はタスク P_i がリリースされてから経過した時間，V_i は t_i を受けて t_i の時点の P_i の価値を返す価値関数）

また，タスクの情報として以下のものが与えられているとする．

C_i　タスク P_i の計算時間

つまりタスクの要素 c_{P^i} は $c_{P^i} = C_i$ から始まってだんだん減っていき，$c_{P^i} = 0$ になったら処理が終了したことを表す．

□

次に拡張時間オートマトンの定義について述べる．正式に定義するため，action と呼ばれるアルファベットの有限集合 Act，clock と呼ばれる実数値の変数の有限集合 C を定義する．a,b,…は Act の要素であり，x_1,x_2,…は C の要素である．$B(C)$ は g などで表され，形式的には clock の制約式の連言で示す．$B(C)$ の要素は clock constraints と呼ばれる．また，m，n は自然数である．

Definition3.1.2 （拡張時間オートマトン）[6]

action の集合 Act とクロックの集合 C，タスクの集合 $\mathbf{P}$ 上のオートマトンは $\langle N, l_0, E, I, M\rangle$ の組である．

- N はノードの有限集合
- $l_0 \in N$ は最初のノード
- $E \subseteq N \times B(C) \times Act \times 2^C \times N$ はエッジの集合
- $I : N \hookrightarrow B(C)$ はノードを clock constraint を割り当てる関数
- $M : N \hookrightarrow P$ はノードにタスクを割り当てる部分関数

エッジに割り当てられた clock constraint を guard，ノードに割り当てられた clock constraint を invariant と呼ぶ．$\langle l, g, a, r, l'\rangle \in E$ のとき $l \xrightarrow{g,a,r} l'$ と書く．また，通常，C から非負の実数への関数（clock assignment と呼ばれる）によって clock の値を示し，clock assignment の集合を ν で示す．

□

拡張時間オートマトンの状態は (l, σ, q) で表す．

l　現在操作しているノード
σ　clock の現在の値を示す clock assignment
q　現在のタスクキュー

タスクキューは OS の待ち行列のようなものでここにタスクが加えられ，先頭から実行され処理が終了するとそのタスクはタスクキューから取り除かれる．

3.1.2 意　味

次に拡張時間オートマトンの semantics について説明していく．まずタスクキューの変化を計算するために次の関数を定義する．

Definition3.1.3 （関数 Sch，関数 Run）[6)]

$Sch(q)$　タスクキューのあるスケジューリングアルゴリズムでソーティングする関数
$Run(q,t)$　実数 t が与えられたときに，時間 t 後のタスクキューを返す関数

□

関数 $Run(q,t)$ は次のような計算を行う．ただし，$q = [P^1(c_{P^1}, V_{P^1}(t_{P^1})), \ldots, P^n(c_{P^n}, V_{P^n}(t_{P^n})]$ である．

○ **すべてのタスクを実行した場合**

$(t \geq c_{P^1} + c_{P^2} + \cdots + c_{P^n})$

$$
\begin{aligned}
&Run(q,t) = \\
&[P^1(0, V_{P^1}(t_{P^1} + c_{P^1})), \\
&P^2(0, V_{P^2}(t_{P^2} + c_{P^1} + c_{P^2})), \\
&\vdots \\
&P^n(0, V_{P^n}(t_{P^n} + c_{P^1} + c_{P^2} + \cdots + c_{P^n}))]
\end{aligned}
$$

○ **タスク P^i まで実行した場合**

$(c_{P^1} + \cdots + c_{P^i} \leq t \leq c_{P^1} + \cdots + c_{P^i} + c_{P_{i+1}})$

$$
\begin{aligned}
&Run(q,t) = \\
&[P^1(0, V_{P^1}(t_{P^1} + c_{P^1})), \\
&\vdots \\
&P^i(0, V_{P^i}(t_{P^i} + c_{P^1} + \cdots + c_{P^i})), \\
&P^{i+1}(c_{P^{i+1}} - (t - (c_{P^1} + \cdots + c_{P^i})), V_{P^{i+1}}(t_{P^{i+1}} + t)), \\
&\vdots \\
&p^n(c_{P^n} - (t - (c_{P^1} + \cdots + c_{P^i})), V_{P^n}(t_{P^n} + t))]
\end{aligned}
$$

ここではタスクをタスクキューの先頭から順に $P^1, P^2, \cdots, P^n$ で表し，$1 \leq i \leq n$ とし，時間経過前のタスク P^i は $P^i(c_{P^i}, V_{P^i}(t_{P^i}))$ で表している．

関数 Sch の行う処理について説明する．従来の（価値関数を用いない，ハードリアルタイムシステムのもの）スケジューリングポリシでは次のような計算をしてタスク P_i をタスク P_k よりタスクキューの前に並べ替える．ハードリアルタイムシステムのものなので，相対デッドライン（タスクがリリースされてからデッドラインまでの時間）が使われている．ここでは，タスク P_i の相対デッドラインを D_i，相対デッドラインまでの時間を d_i で表している．

- **Highest priority first（FPS）：**
  $P_i \in q,\ \forall P_k \in q\ Pr(P_i) \geq Pr(P_k)$
  ここで $Pr(P)$ は P の固定優先度を示す
- **Earliest deadline first（EDF）：**
  $P_i \in q,\ \forall P_k \in q\ d_i \leq d_k$
- **First come first served（FCFS）：**
  $P_i \in q,\ \forall P_k \in q\ D_i - d_i \geq D_k - d_k$
- **Least laxity first（LLF）：**
  $P_i \in q,\ \forall P_k \in q\ d_i - c_i \leq d_k - c_k$

しかし，HVF や HVDF ような価値関数を用いた場合，時間経過中にタスクの優先度の高低が入れ替わることがある．後述するが，従来のアルゴリズムを用いた解析と同じく，本研究もタスクが新しく加えられたときに関数 Sch でタスクキューを並べ替えるようにしている．そこで，まず実行中のタスクが新しく加わったタスク以外にプリエンプトされることのないように，前述のとおり，価値関数を単調減少の関数のみとする．単調減少でないタスクの例とは図 4 の V4 のようなデッドライン付近で最も価値が高くなるような価値関数である．このようなタスクは特別な例であるため今回は対象外とする．また，タスク実行中（時間経過中）に次に優先度の高いタスクが変わることを考えて HVF と HVDF の関数 Sch は次のようにする．n はタスクキュー q の中のタスクの数とする．

HVF　・$n = 0, 1$ の場合：何もしない

・$n \geq 2$ の場合：最も価値の高いタスク P_i を q の先頭にもってきて，残りのタスク P_j，P_k などを次のようにする．

$P_i \in q, \forall P_k \in q$

$V_i(t_i + c_i) \geq V_k(t_k + c_k)$

$Sch([P_j(c_j - c_i, V_j(t_j + c_i)),$

$\quad P_k(c_k - c_i, V_k(t_k + c_i)), \cdots])$

HVDF ・$n = 0, 1$ の場合：何もしない

・$n \geq 2$ の場合：最も価値の高いタスク P_i を q の先頭にもってきて，残りのタスク P_j，P_k などを次のようにする．

$P_i \in q, \forall P_k \in q$

$V_i(t_i + c_i)/c_i \geq V_k(t_k + c_k)/c_k$

$Sch([P_j(c_j - c_i, V_j(t_j + c_i)),$

$\quad P_k(c_k - c_i, V_k(t_k + c_i)), \cdots])$

このように，実行中のタスクの処理後の状態のタスクを考えて，タスクを再帰的に並べ替えていくことにより，タスクが新しく加えられたときのみ関数 Sch でタスクキューを並べ替えればよいようにしている．この関数を用いて拡張時間オートマトンの semantics を定義する．

Definition3.1.4 （意味）

スケジューリング関数 Sch が与えられたときの，初期状態 (l_0, σ_0, q_0) のタスク付き時間オートマトン $\langle N, l_0, E, I, M\rangle$ の遷移体系を次のように定義する．

- $(l, \sigma, q) \xrightarrow{a} (m, \sigma[r \mapsto 0], Sch(M(m) :: q))$
  if $l \xrightarrow{g,a,r} m$ and $\sigma \models g$ （離散遷移）
- $(l, \sigma, q) \xrightarrow{t} (l, \sigma + t, Run(q, t))$
  if $\forall\ 0 \leq t\prime \leq t\ .\ (\sigma + t\prime) \models I(l)$（遅延遷移）

d は非負の実数であり，$\sigma + d$ は clock の集合 C の各 clcok x を値 $\sigma(x) + d$ に写像する clock assignment である．$r \subseteq C$ であり，$\sigma[r \mapsto 0]$ は r の各 clock を値 0 に写像する C の assignment であり，r を除く C 上の σ と一致する．$\sigma \models g$ であるなら clock assignment σ は制限 g を満たすことになる．ここで，$M(m) :: q$ とは $M(m)$ を q に挿入したタスクキューである．

□

3.1.3 動 作 例

図 13 の動作例の 1 つを示す．

$(m, [x = 0, y = 0][])$

$\xrightarrow{3} (m, [x = 3, y = 3], [])$

$\xrightarrow{a} (n, [x = 0, y = 3], [P_2(2, V_2(0))])$

$\xrightarrow{a} (n, [x = 0, y = 0], [P_2(2, V_2(0)), P_2(2, V_2(0))])$

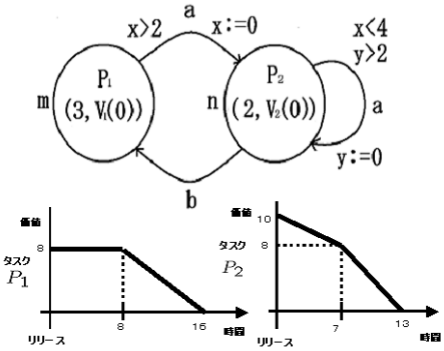

図 13 拡張時間オートマトンの例 1

Fig. 13 Example 1 of extended automaton.

$\xrightarrow{3} (n, [x = 3, y = 3], [P_2(1, V_2(3))])$

$\xrightarrow{a} (n, [x = 3, y = 0], [P_2(2, V_2(0)), P_2(1, V_2(3))])$

$\xrightarrow{1} (n, [x = 4, y = 1], [P_2(1, V_2(1)), P_2(1, V_2(4))])$

$\xrightarrow{b} (m, [x = 4, y = 1], [P_2(1, V_2(1)),$

$\quad P_2(1, V_2(4)), P_1(3, V_1(0))])$

$\vdots$

まず最初にノード m で 3 の遅延遷移が起こり時間が経過する．時間が経過するとリリースされてからの経過時間 t_i が加算されていく．次にノード n への離散遷移が起こりタスクキューにタスク P_2 が加えられ，続いて同様に離散遷移が起こりタスクキューにタスク P_2 が加えられる．そして 3 の遅延遷移が起こると時間が経過するのでタスクキューの先頭のタスクが実行され終了し，タスクキューから取り除かれる．また，5 行目では HVDF の方針に従って value density の高い新しく来たタスク P_2 がタスクキューの先頭に並べ替えられている．このように拡張時間オートマトンは動作していく．

3.2 解 析 手 法

次に，解析手法について説明する．我々の以前の解析方法[6] を，タスクの順序制約およびリソース制約が扱えるように拡張した．この拡張により，より現実的なシステムの解析が可能となる．

まず，拡張時間オートマトン A とスケジューリングポリシ Sch が与えられたとき，2 つの時間オートマトン $E(Sch)$ と $E(A)$ を生成する．この $E(Sch)$ はタスクキューとスケジューリングポリシ Sch，タスクの制約をオートマトンで表現したものとし，スケジューラオートマトンと呼ぶことにする．また，$E(A)$ はタスク P_i が割り当てられたノードへの遷移に，$release_i$ をラベル付けした時間オートマトン（拡張時間オートマトンではない）となる．この 2 つの時間オートマト

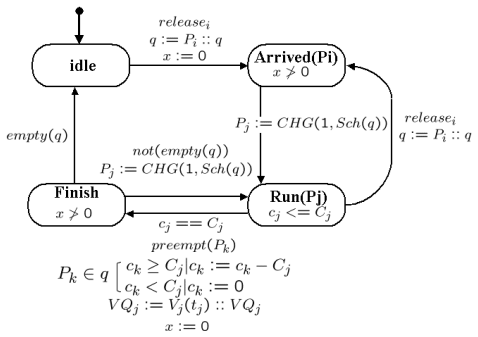

図 14 スケジューラオートマトン
Fig. 14 Scheduler automaton.

ンを並列合成（同期積）したものの動作を調べることによって，性能解析を行う．

また，Yi らの解析手法[3] との違いは実行価値の計算が行われている点，リソース制御に加えて，タスクがメッセージ通信を行っている点である．

3.2.1 スケジューラオートマトン

スケジューラオートマトンの要素を定義していく．

Definition3.2.1 （スケジューラオートマトンの clock）

c_i タスク P_i の処理時間を測る clock.

t_i タスク P_i がリリースされてからの時間を測る clock.

c_i，t_i ともにタスクの数だけあるとする．

□

Definition3.2.2 （タスクキュー）

$empty(q)$ タスクキュー q が空であることを表す.

$q := P_i :: q$ タスクキュー q にタスク P_i を加える.

$preempt(P_i)$ タスクキュー q からタスク P_i を取り除くことを表す.

$CHG(h, q)$ タスクキュー q の h 番目のタスクを表す.

□

Definition3.2.3 （実行価値保存のための配列）

VQ_i タスク P_i の実行価値（実行終了時の価値関数の値）を保存するための配列.

$VQ_i := V_i(t_i) :: VQ_i$ で配列 VQ_i に $V_i(t_i)$ を加えることを表す.

□

タスクキューとスケジューリングポリシ Sch をオートマトンとして表現した $E(Sch)$ を**図 14** に示す．図 14 は，上記どおり，タスクキューとスケジューリングポリシ Sch をオートマトンで表現したもので，タスクの制約を含んでいない．なお，タスクの制約とは，リソースへの排他制御とタスク間の同期通信，非同期通信であり，リソースへの排他制御はセマフォアを使って実現し，タスク間の同期通信と非同期通信はバッファを使って実現する．図 14 では，$idle$ はタスクキューが空の状態を表し，$Arrived(P_i)$ はタスク P_i が到着している状態を表し，$Run(P_j)$ は P_j が実行中の状態を表し，$Finish$ はタスクの実行が終了した状態を表している．

リソースへの排他制御の実現：

次に，セマフォアを使った，リソースへの排他制御の実現を示す．

Definition3.2.4 （セマフォア）

次の 3 つはタスク P_i のセマフォア s へのアクセスパターンを示すリストを表す配列である．

$times_i$ タスク P_i がセマフォアを lock/unlock する時点を示しており，最後の要素は実行時間 C_i.

sem_i タスク P_i に関連するセマフォアの状態を表し，最後の要素はつねに NONE であるとする．

$locker$ 整数配列であり，$locker[s]$ で現在，セマフォア s を lock しているタスクの index を表す．もし s が lock されていないならば $locker[s]$ の値は NONE になる．

セマフォア表現に用いるカウンタを定義する．

k_i $times_i[k_i]$，$sem_i[k_i]$ で配列 $times_i$，sem_i の要素を表す，タスク P_i のカウンタ

これらは static であり，システム記述の一部である．

□

排他制御の際に問題となる優先度逆転を防ぐために，Highest Locker Protocol（HLP）を本手法にあったものに変更したものを用いる．HLP とは，そのリソースを用いるすべてのタスクの中で最も高い優先度（ceiling 優先度）を，リソースを lock している間は継承するというものである．単純な手法であり，実装が簡単であるため多くのシステムで用いられている．本研究ではタスクの優先度は HVDF で決まるため優先度が動的に変化する．そこで，ceiling 優先度として，リソースを用いるすべてのタスクの時間経過 0 での value density が最も高いものを用いる．HLP を実現するのに必要な次のものを定義する．

Definition3.2.5（Highest Locker Protocol（HLP））

$ceil(s)$ セマフォア s の ceiling 優先度を返す関数

$prio_i$ 排他制御によって変更されたタスク P_i の優先度

□

図 14 のスケジューラオートマトンにリソース制限を追加したものを**図 15** に示す．図 15 は図 14 から

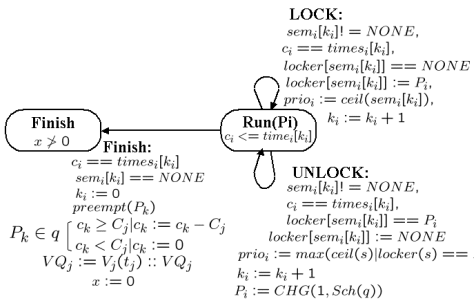

図 15 スケジューラオートマトン（リソース制御）
Fig. 15 Scheduler automaton (resource control).

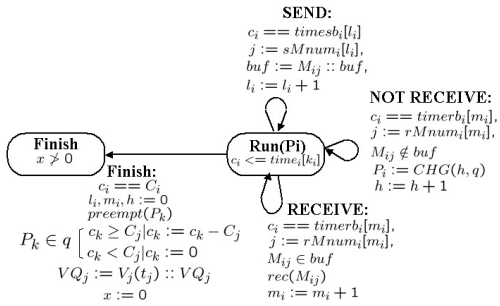

図 16 スケジューラオートマトン（メッセージ通信）
Fig. 16 Scheduler automaton (message communication).

の変化した部分だけを示したものである．図 14 では，状態 $Run(P_i)$ において，タスク P_i が実行中に，セマフォアを使って，リソースを LOCK か UNLOCK することを実現する．

メッセージ通信の実現：

次に，バッファを実現する手法を説明する．まず，準備として，すべてのメッセージには，メッセージを一意に特定できるナンバリング（メッセージナンバ）がされているものとする．

Definition3.2.6 （バッファ）

バッファを表現するため次のものを定義する．

$timesb_i$ タスク P_i がバッファに対してメッセージを送信する時点を示している．

$timerb_i$ タスク P_i がバッファに対してメッセージを受信要求する時点を示している．

buf 非同期メッセージを保存しておくためのシステムのバッファ．

M_{ij} タスク P_i のメッセージナンバ j のメッセージ．

$sMnum_i$ タスク P_i の送信するメッセージナンバを示した配列．

$rMnum_i$ タスク P_i の受信要求するメッセージナンバを示した配列．

buf への操作として，$buf := M_{ij} :: buf$ でバッファにメッセージ M_{ij} を送り，$rec(M_{ij})$ でメッセージ M_{ij} をバッファから取り除くとする．

バッファ表現に用いるカウンタを定義する．

l_i $timesb_i[l_i], sMnum_i[l_i]$ で $timesb_i$，$sMnum_i$ の要素を表すカウンタ．

m_i $timerb_i[m_i], rMnum_i[m_i]$で$timerb_i$, $rMnum_i$ の要素を表すカウンタ．

これらは static であり，システム記述の一部である．

□

図 15 のスケジューラオートマトンにメッセージ通信を追加したものを**図 16** に示す．図 16 は図 15 からの変化した部分だけを示したものである．図 15 では，状態 $Run(P_i)$ において，タスク P_i が実行中に，バッファを使って，メッセージを送信か受信か受信不能を実現する．

3.2.2 解析方法

解析は $E(Sch)$ と $E(A)$ を並列合成してできた時間オートマトンの動作列を調べることによって，性能解析を行う．解析は時間オートマトン $E(A)$ の全状態を調べる．時間オートマトンの状態は (l, σ, q) の組合せであるので，有限の状態で表現できる．なぜならば，ノードは定義から有限であり，clock も拡張時間オートマトン A には減算がないので同様に有限である．また，タスクキューも前述の前提条件から有限であるからである．そして，各タスクごとの実行価値を調べ，その最高値，最低値を調べる．時間オートマトンの解析は，一般的に用いられている，ゾーンオートマトンを用いて行う．並列合成手法，ゾーンオートマトンを用いた解析手法について詳しくは文献 6)，18) に示されている．

4. 事例研究

本章では，文献 16) で採用している，メディアサーバアプリケーションを例として，本手法を適用する．今回用いるメディアサーバアプリケーションのクラス図は**図 17** のようになっている．これは，デジタルテレビ（DT）とオーディオシステム（AS）がメディアサーバに対してデータ取得の要求を出し，メディアサーバが必要なデータ返信するといったアプリケーションである．

4.1 タスクマッピング

図 17 のクラス図を，タスク構造化基準 (1) I/O Task Structuring Criteria，(2) Internal Task Structuring Criteria を適用し，タスク候補となるアクティブオブジェクトを選出する．初期タスクマッピングを行ったメディアサーバアプリケーションのオブジェクト図は**図 18** のようになる．太線で囲まれたオブジェクトが

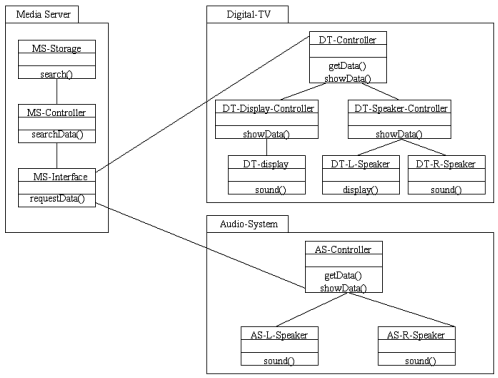

図 17 メディアサーバアプリケーションのクラス図
Fig. 17 Class diagram of media server application.

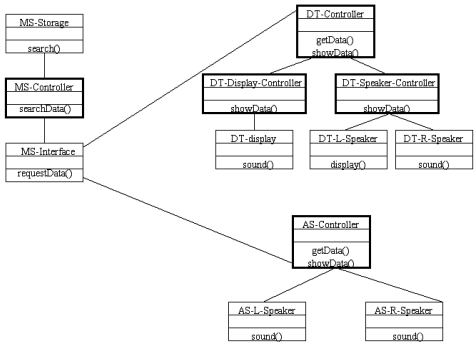

図 18 メディアサーバアプリケーションのオブジェクト図（初期タスクマッピング）
Fig. 18 Object diagram of media server application (initial task mapping).

タスク候補である．

次に，タスク構造化基準 (4) Task Clustering Criteria を基に，再タスクマッピングを行う．Temporal Clustering によってタスク候補 DT-Display-Controller と DT-Speaker-Controller がグループ化できる．また，そのまとめたタスク候補と DT-Controller が Sequential Clustering によって 1 つの物理タスクにグループ化できる．再タスクマッピングを行ったオブジェクト図を**図 19** に示す．

4.2 タスク構造図の生成

次にタスク構造図を生成する．メディアサーバアプリケーションのシーケンス図（タスクマッピング後）を**図 20**，**図 21** に示す．タスク DT-Controller は複数のタスクをグループ化したものであるため価値関数の形が複雑になっている．シーケンス図，オブジェクト図より生成したタスク構造図は**図 22** のようになる．

4.3 性 能 解 析

図 22 のタスク構造図より，拡張時間オートマトンを生成すると**図 23** のようになる．図 22 では，タスク

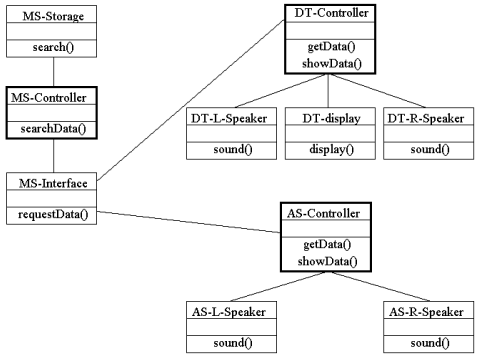

図 19 メディアサーバアプリケーションのオブジェクト図（再タスクマッピング）
Fig. 19 Object diagram of media server application (second task mapping).

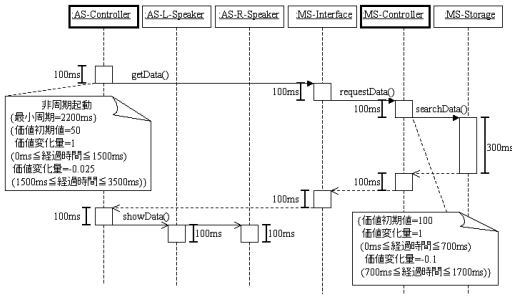

図 20 メディアサーバアプリケーションのシーケンス図 (1)
Fig. 20 Sequence diagram of media server application (1).

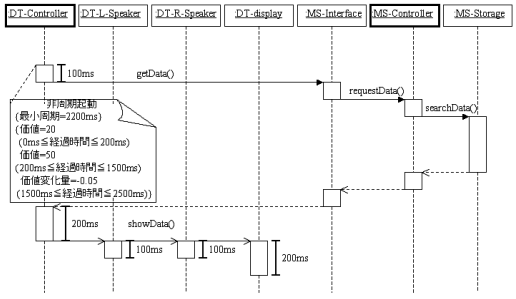

図 21 メディアサーバアプリケーションのシーケンス図 (2)
Fig. 21 Sequence diagram of media server application (2).

クは DT-Controller，MS-Controller，AS-Controller の 3 つであり，各タスクが図 23 の各ノードに対応する．また，タスクの残り計算時間と現在の価値はシーケンス図のノートのタグ付き値などを用いて決める．たとえば，タスク MS-Controller に対応するノード MS では，残り計算時間は 700 で，価値関数は V_1 である．また，離散遷移にともなうガードはタスクの最小周期などから決まる．

図 23 の拡張時間オートマトンから**図 24** の時間オートマトンを構成して，この時間オートマトンと 3 章の

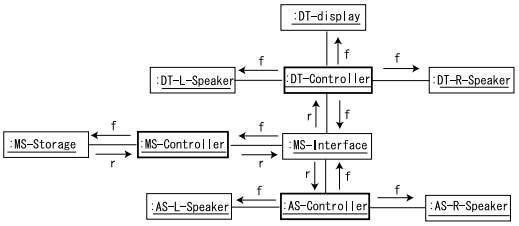

図 22 メディアサーバアプリケーションのタスク構造図
Fig. 22 Task structure diagram of media server application.

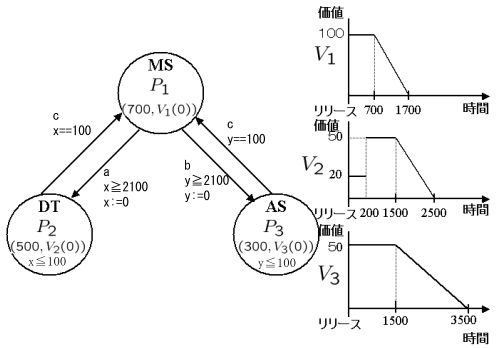

図 23 メディアサーバアプリケーションの拡張時間オートマトン
Fig. 23 Extended timed automaton of media server application.

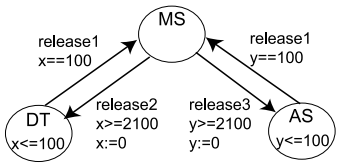

図 24 メディアサーバアプリケーションの時間オートマトン
Fig. 24 Timed automaton of media server application.

表 2 価値関数の値の最大値と最小値
Table 2 Maximum values and minimum values of utility function.

	最大値	最小値
P_1	$V_1(700)$	$V_1(700)$
P_2	$V_2(1200)$	$V_2(2200)$
P_3	$V_3(1000)$	$V_3(2200)$

図 14 のスケジューラオートマトンとを並列合成して解析した結果，タスクの実行価値の最大値と最小値は以下のようになった．タスク P_2，P_3 はタスク P_1 と同期メッセージ通信を行っているので，最短でもタスク P_1 の実行を待つことになる．また，タスク P_1 は価値関数の値が高いのでプリエンプトされることがなく実行価値の最大値と最小値が変わらない（**表 2**）．

以上のように，形式的手法により，設計段階でタスクの価値の最大値と最小値を計算できることは非常に有効であり，既存研究には存在しない．

4.4 議　　論

本稿と大阪大学の岡野や谷口らの研究[16] とには，以下の違いがある．

(1) 本稿では，ソフトリアルタイムシステムのタスクの設計と価値解析を対象として，UML からソフトリアルタイムなタスク構造の設計およびタスクの終了時刻の解析による価値の最大値と最小値の計算を実現している．

(2) 大阪大学の岡野や谷口らの研究[16] では，ハードリアルタイムシステムの分散実時間アプリケーションのコンポーネント設計を対象として，線形制約式判定問題によるコンポーネント間のタイミング制約の充足性判定およびモデル検査によるコンポーネント内部のタイミング制約の充足性判定を実現している．

また，本稿の研究において，タイミング制約の充足性やデッドロックなどの論理的な振舞いの検証に関しては，スケジューリングに関する情報をエンコードした時間オートマトンと，リソース制御などをエンコードしたスケジューラオートマトンとの並列合成で得られたオートマトンを到達可能解析すれば，検証できる．

5. 結　　論

今回，ソフトリアルタイムシステムの性質を価値関数で表現したものを，UML で記述し，タスク構造化を行い，拡張した時間オートマトンに変換して形式的検証手法による性能解析を行う方法を示した．本手法によって，UML での設計段階からソフトリアルタイムシステムの信頼性を保証することができる．今後の課題としては，大規模なシステムを検証できるよう解析コストを削減する手法を考えること，および，より現実的な問題への本手法を適用することなどがある．

謝辞 有益なコメントをいただいた査読者各位に感謝します．本研究は文部省科研費基盤 C（課題番号 19500025）の援助の下で実施されました．

参 考 文 献

1) Fidge, C.J.: Real-time scheduling theory, SVRC Services (UniQuest Pty Ltd) Consultancy Report 0036-2 (2002).
2) Norstrom, C., Wall, A. and Yi, W.: Timed Automata as Task Models for Event-Driven Systems, *RTCSA*, pp.182–189, IEEE CS (1999).
3) Fersman, E. and Yi, W.: A Generic Approach to Schedulability Analysis of Real Time Tasks, *Nordic Journal of Computing*, Vol.11, No.2,

pp.129–147 (2004).
4) Jensen, E.D., Locke, C.D. and Tokuda, H.: A Time-Driven Scheduling Model for Real-Time Operating Systems, *IEEE Real-Time Systems Symposium*, pp.112–122 (1985).
5) Gardner, M.K. and Liu, J.W.S.: Analyzing Stochastic Fixed-Priority Real-Time Systems, LNCS 1579, pp.44–58 (1999).
6) 山根　智：時間オートマトンによるリアルタイムシステムの性能解析，情報処理学会論文誌，vol.46, No.11, pp.2410–2421 (2005).
7) Ronen, Y., Mosse, D. and Pollack, M.E.: Value-Density Algorithms for the Deliberation-Scheduling Problem, *SIGART Bulletin*, Vol.7, No.2, pp.41–49, ACM (1996).
8) Shlaer, S. and Mellor, S.J.: *Object Lifecycles: Modeling the World in States*, Yourdon Press Computing Series (1991).
9) Selic, B., Gullekson, G. and Ward, P.T.: *Real-Time Object-Oriented Modeling*, Wiley Professional Computing, John Wiley Sons Inc. (Computers) (1994).
10) Awad, M., Kuusela, J. and Ziegler, J.: *Object-Oriented Technology for Real-Time Systems: A. Practical Approach Using Omt and Fusion*, Prentice Hall (1996).
11) OMG. http://www.uml.org/
12) Gomaa, H.: *Designing Concurrent, Distributed and Real-Time Applications with UML*, Addison-Wesley (2000).
13) OMG: UML Profile for schedulability, performance and time specification. Version 1.0, formal/03-09-01 (2003).
14) Douglass, B.P.: *Real Time Uml 3th edition Advances in the Uml for Real-Time Systems*, Addison-Wesley Object Technology Series (2004).
15) Kim, S., Park, J. and Hong, S.: Scenario-Based multitasking for real-time object-Oriented models, *Journal of Information and Software Technology*, Vol.48, pp.820–835 (2006).
16) 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, pp.683–692 (2006).
17) Pilone, D. and Pitman, N.: UML2.0 クイックリファレンス，O'REILLY (2006).
18) Alur, R.: Timed Automata, LNCS 1633, pp.8–22 (1999).

(平成 19 年 2 月 9 日受付)
(平成 19 年 6 月 5 日採録)

坂倉　賢昭

2007 年 3 月金沢大学大学院自然科学研究科博士課程前期修了．同年京セラ（株）入社．現在に至る．在学中，UML によるソフトリアルタイムシステムの設計解析手法の研究に従事．

山根　　智（正会員）

1984 年 3 月京都大学大学院修了．現在，金沢大学大学院自然科学研究科電子情報科学専攻教授．リアルタイムシステムの形式的検証の研究に従事．電子情報通信学会，ソフトウェア科学会，EATCS 等各会員．