卒 業 論 文

実時間並行ソフトウェアの開発支援ツールの構築

A Tool for supporting development of Real-Time Concurrent Software

名古屋大学工学部電気電子・情報工学科
情報工学コース

桑 原 寛 明

平成 12 年度

指導教官　　阿草 清滋　教授

概要

本研究では時間付きデザインパターンに基づく実時間並行ソフトウェア開発を支援するツールの設計及び実装を行い，時間付きデザインパターンに基づく開発の支援環境を構築する．

時間付きデザインパターンは時間制約に注目して従来のデザインパターンを拡張し，実時間並行ソフトウェアのパターン化とその再利用を容易にする．時間付きデザインパターンに基づく開発では，実時間並行ソフトウェアを時間オートマトンにより拡張したUMLを用いてモデル化し，テンプレートコードを作成する．ソフトウェアの時間制約を時間オートマトンを用いて表現し，時間制約が満たされるかを確認する．

時間付きデザインパターンに基づく開発向けの統合環境を提供することで開発プロセスの支援を行う．ツールを用いることで時間付きデザインパターンを図のまま扱い開発を進めることができる．

abstract

This thesis presents a development tool for real-time concurrent software based on a new design principle, called ”timed design patterns”.

Timed design patterns extend conventional design patterns focusing on time constraints in order to patternize and reuse real-time concurrent software. Throughout a development process real-time concurrent software is modeled by UML-like documents extended with Alur's timed automata in the course of generating template source codes. Timed automata characterize the timed property of the real-time concurrent software in an abstract way in order to ensure the time constraints.

Our tool eases the development process by providing an integrated environment for timed design patterns. The environment enables to manipulate timed design patterns through the visual interface.

目次

第 1 章

はじめに

オブジェクト指向設計に基づいたソフトウェア開発では、ソフトウェアを部品化することによって効率の良い開発を行うことができる。デザインパターン [3] はオブジェクト指向設計及び分析の際に現れるプログラムの典型的なパターンを分類し、ソフトウェア部品や設計の再利用を容易にする。そこで実時間並行ソフトウェアに対してもデザインパターンを適用して再利用したいという要求がある。

オブジェクト指向設計及び分析では、ソフトウェアの制御構造やデータ構造といった静的な構造はクラス図を用いて表し、ソフトウェアの動的な振舞いはシーケンス図やコラボレーション図を用いて表す。デザインパターンでもこれらの図を用いてパターンを表現している。従来のデザインパターンではソフトウェアの動的な振舞いの性質である時間制約を、静的構造を表すクラス図ではなく、動的振舞いを表すシーケンス図やコラボレーション図に制約を付加することで表現する。このように時間制約をソフトウェアの構造と独立して表現するため、時間制約が部品化に直接反映されないという問題がある。実時間並行ソフトウェアは並行性と実時間性から実行のトレースが常に同じになるとは限らないが、シーケンス図によって表されるのは実行の一つのトレースのみである。

時間制約をソフトウェアの静的構造と併せて表現すれば時間制約を部品化に反映でき、再利用が容易になる。そこで、デザインパターンを時間制約に注目して拡張した時間付きデザインパターン [1] と、時間付きデザインパターンに基づく実時間並行ソフトウェアの開発手法 [2] が提案されている。

時間付きデザインパターンではソフトウェアの動的な振舞いである時間制約を時間オートマトンを用いて表し、その時間オートマトンをクラス図に記述する。このように動的な振舞いを静的な構造と一体化して表現することでソフトウェアの部品化に反映されるようにする。文献 [2] で提案されている開発手法では、クラス図に記述された時間オートマトンそれぞれを実時間スレッドにラップすることで実現する。開発は時間オートマトンが与えられた仕様を満たすかどうかを時間オートマトンの検証系である UPPAAL[4, 5] を用いて検証しながら進めるため、最終的に時間制約が守られることが保証される。

本研究では時間付きデザインパターンに基づく実時間並行ソフトウェアの開発を支援するツールの設計及び実装を行い、時間付きデザインパターンに基づく開発の支援環境を構築する。時間付きデザインパターンに基づく開発では、時間オートマトンの検証やクラス図の詳細化を既存のプログラムを用いて行う。支援ツールでは既存プログラムとのインターフェースを提供することで、既存プログラムの詳細を意識する必要のない開発を可能にし、時間付きデザインパターンに基づく開発を全面的に支援する。支援ツールの実装には Java を用いる。

本論文では、まず第 2章で時間付きデザインパターンに基づく開発手法を説明し、どのようにツールで支援するか述べる。第 3章で開発支援ツールの機能やシステム構成について説明する。第 4章で例を示し、最後にまとめと今後の課題を述べる。付録に本研究で作成した支援ツールのマニュアルとソースを添えた。

第 2 章

時間付きデザインパターンに基づくソフトウェア開発手法

2.1 時間付きデザインパターン

2.1.1 概要

時間付きデザインパターン [1] は従来のデザインパターンを時間制約に注目して拡張する。従来のデザインパターンを適用した実時間並行ソフトウェアの設計・開発では、時間制約は実装上では存在するがパターンには含まれない。時間付きデザインパターンではクラスのメソッドコードの記述の拡張として時間オートマトンの記述を許す。時間オートマトンを用いて時間制約をクラス図に記述し、時間制約まで含めたデザインパターンを作成する。時間オートマトンを用いて振舞いを表現することで、実時間並行ソフトウェアの実行はその並行制御の非決定性から一意ではないにもかかわらずそのうちの一つの実行のトレースしか表現できないシーケンス図の問題を克服する。

ここでは、実時間並行ソフトウェアは複数の制御フローを持ち、それらのフロー間で通信を行うことで計算を進めると捉える。この場合制御フロー間の同期イベント通信がソフトウェアの振舞いを決定する。しかし並行性と実時間性により非決定性が存在するためソフトウェアの振舞いは一通りには決まらない。このようなソフトウェアが時間制約を満たすか調べるには実行の全ての組合せ

についてチェックを行う必要がある。制御フローを時間オートマトンを用いて表し、チェックを時間オートマトンの検証系である UPPAAL[4, 5] を用いて行う。

時間付きデザインパターンに基づく実時間並行ソフトウェア開発手法 [2] では時間オートマトンを実時間スレッドによってラップし、スレッドに対する時間制約によってソフトウェアの時間制約を満たす。時間オートマトンに対する仕様は設計の段階から与え、時間オートマトンが仕様を満たすかどうかを検証しながら開発を進めていく。こうすることで時間制約が守られることを常に保証できる。

2.1.2 表記法

時間付きデザインパターンのクラス図の表記には UML[6] を用いる。UML(Unified Modeling Language) とは、オブジェクト指向設計・分析においてシステムをモデル化する際の表記法と用語を規定するモデリング言語である。

時間付きデザインパターン独自の要素として以下の要素を導入する。

Timed Behavior インターフェース

時間制約が存在するクラスはこのインターフェースを実装する。時間制約が存在することを示すための印に用いられるインターフェースであり、メソッド定義は持たない。

timed behavior ステレオタイプ

Timed Behavior インターフェースを実装するクラスの振舞いの記述を表す。振舞いを表現する時間オートマトン及びイベントとメソッドの対応関係が記述される。通常はノートが timed behavior ステレオタイプ化され、時間制約が存在するクラスと対応付けられて表示される。

timed design pattern ステレオタイプ

≪timed behavior≫ [1]間のコラボレーションを表す。時間オートマトンをラップする実時間スレッド間で送受信される同期イベントについて記述する。

時間付きデザインパターンの表記例を図 2.1に示す。

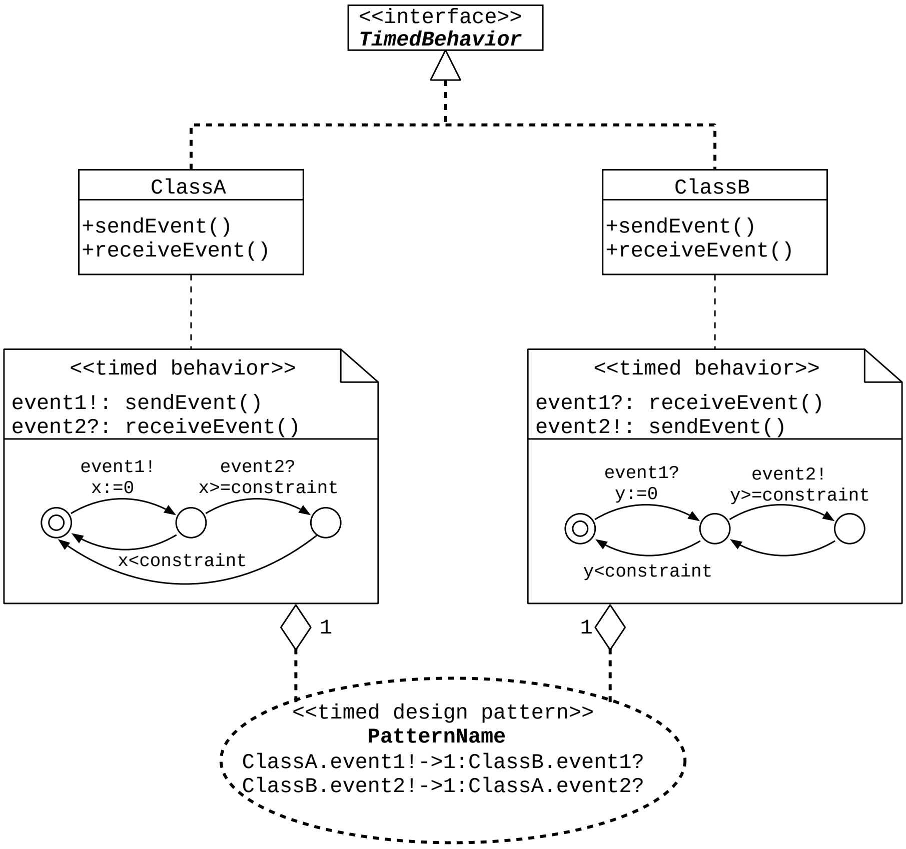

図 2.1: 時間付きデザインパターンの表記例

ソフトウェアのオブジェクト指向設計・分析では、ソフトウェアの静的な構造、特にクラスのような要素や、その内部構造、他の要素との関連についてク

[1]UML では stereotype ステレオタイプを ≪stereotype≫ と表記する

		従来のデザインパターン	時間付きデザインパターン
静的構造		クラス図	クラス図
動的振舞	相互作用	シーケンス図 コラボレーション図	クラス図内の 時間オートマトン
	状態遷移	ステートチャート図	

表 2.1: 設計時に用いる図とその表す内容

ラス図を用いて示す。動的な部分であるオブジェクト間の相互作用のパターンについてはシーケンス図とコラボレーション図を用いて示す。シーケンス図はオブジェクト間の時系列に沿った相互作用、特に相互作用に関わるオブジェクトと時間シーケンスにおいて整列されたオブジェクトが交換するメッセージについて表す。一方コラボレーション図はオブジェクト間の相互作用の全体構造について表す。同じく動的な部分であるオブジェクトの状態と状態間の遷移についてはステートチャートを用いて示す。これらの図とその表す内容について表 2.1にまとめる。

一方、時間付きデザインパターンではクラス図のみでソフトウェアの構造を表す。静的な構造は従来と同様にクラス図を用いて示す。動的な振舞いは時間付きデザインパターンにおいて拡張された時間オートマトンによって表す。時間オートマトンは状態と状態間の遷移であるのでステートチャートと同等の意味を持つ。またシーケンス図やコラボレーション図はそれに制約を付加することで時間制約を記述することができる。しかしこれらの図で表現されるのは、時系列に沿って固定された相互作用によって表されるソフトウェアの実行の一つのトレースに過ぎず、非決定性を持つ並行制御によってその実行が一意に定まらない実時間並行ソフトウェアの動的な振舞いの記述には適さない。一方それ自身が時間制約を記述する時間オートマトンは、それによって表現される振舞いを合成することで並行実行されるオブジェクト間の同期を表し、ソフトウェ

アの任意の実行を記述できる。

2.2 開発手順の概要

時間付きデザインパターンに基づく実時間並行ソフトウェア開発の流れを図2.2に示す。

開発ではまず時間付きデザインパターンに基づいて実時間並行ソフトウェアをその仕様に従って設計を行い、クラス図とクラス図に示される時間オートマトンの満たすべき仕様を作成する。この段階で時間オートマトンとその仕様が得られているので、クラス図に記述されている時間オートマトンが仕様を満たすか検証する。ここが図 2.2の (1) である。

時間オートマトンの検証は、 UPPAAL の検証プログラム verifyta[5] を用いて行う。それに伴って設計の際に生成される振舞いの仕様は UPPAAL の論理式に従って記述する。

2.1節で述べたように実行時環境として時間オートマトンは実時間スレッドにラップする。この時、実時間スレッドの実行時環境に以下の制約を持たせる。

- イベントの送受信に共有キューを用いる
- スレッドのスケジューリングは周期的であるとする

この実行時環境において時間オートマトンによって表される振舞いをスレッドに割り当てるため、スケジューリング情報を与えて時間オートマトンを詳細化し、詳細化されたクラス図を生成する。時間オートマトンのイベントの同期を等価ではない共有キューを用いたイベントの送受信でシミュレートするため、

- キューに保持されるイベントは最大一つ
- イベントの送受信の遅延の時間制約

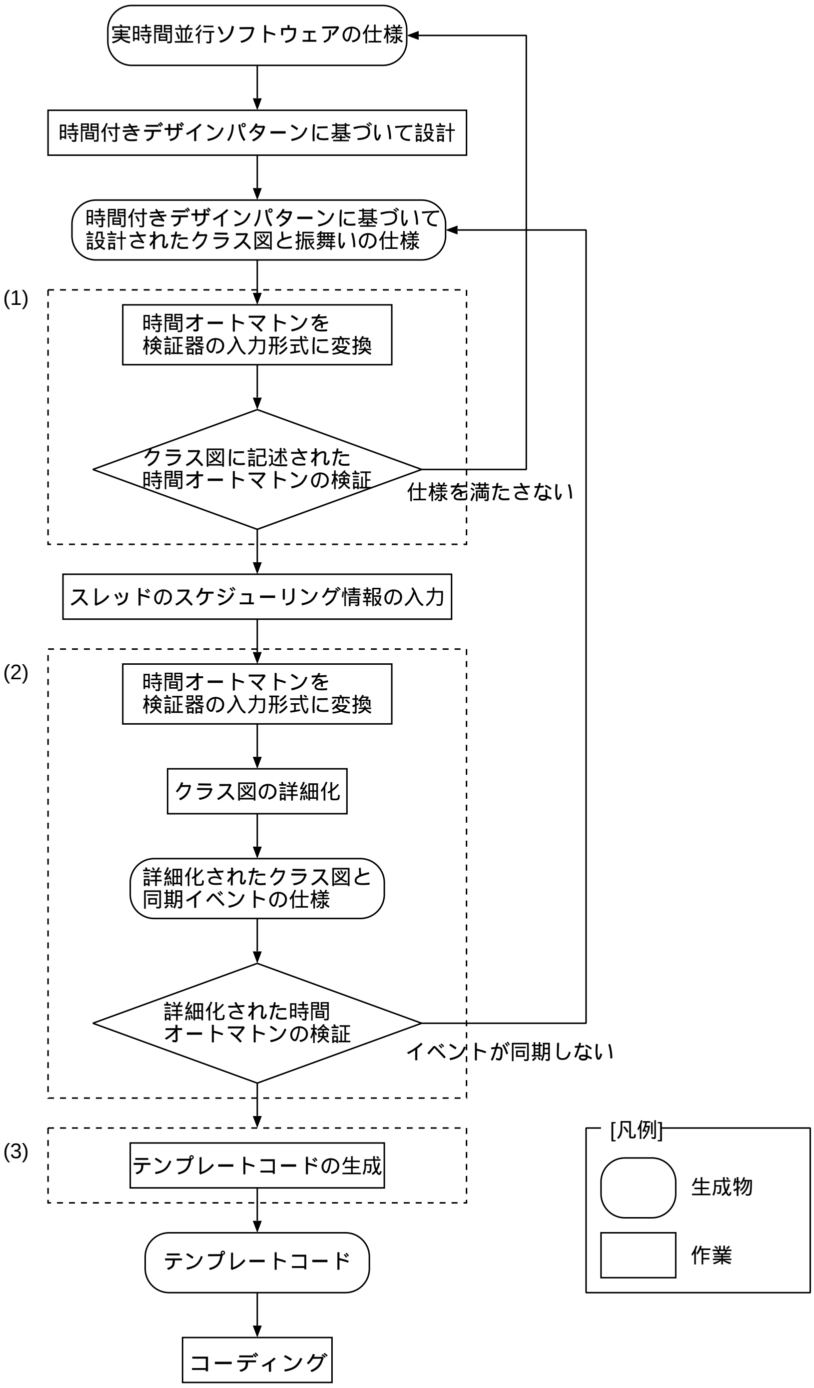
実時間並行ソフトウェアの仕様
時間付きデザインパターンに基づいて設計
時間付きデザインパターンに基づいて
設計されたクラス図と振舞いの仕様
(1)
時間オートマトンを
検証器の入力形式に変換
クラス図に記述された
時間オートマトンの検証
仕様を満たさない
スレッドのスケジューリング情報の入力
(2)
時間オートマトンを
検証器の入力形式に変換
クラス図の詳細化
詳細化されたクラス図と
同期イベントの仕様
詳細化された時間
オートマトンの検証
イベントが同期しない
(3)
テンプレートコードの生成
テンプレートコード
コーディング
[凡例]
生成物
作業

図 2.2: 開発の流れ

という制約が発生する。詳細化された時間オートマトンが初めに与えられた仕様と、詳細化に際して新たに得られたイベント同期の制約の双方を満たすか検証する。ここが図 2.2の (2) に対応する。

最後に、詳細化されたクラス図からテンプレートコードを生成する。生成するコードは実時間並行ソフトウェアを実装するプラットフォームに依存する。この作業が図 2.2の (3) である。

時間付きデザインパターンに基づく開発は、時間オートマトンで表されるソフトウェアの動的な振舞いの仕様が満たされることを手順の各段階で検証しながら進める。そのため最終的に生成されるテンプレートコードに対して時間制約に関係しない部分のコーディングを行えば、ソフトウェアの時間制約は守られている。

2.3 ツールによる支援

時間付きデザインパターンに基づくソフトウェア開発に際して、開発手順中の時間オートマトンの検証、クラス図の詳細化、テンプレートコードの生成の 3 つの作業を行うプログラムがすでに存在する。これらのプログラムを統合するツールによって開発の効率化・省力化を図るとともに、シームレスな開発作業環境を提供する。

開発では初めに時間付きデザインパターンに基づいて設計を行いクラス図と振舞いの仕様を作成する。ここで作られるクラス図とクラス図に含まれる時間オートマトンは通常図として紙や既存のモデリングツール上に描かれる。その後時間オートマトンが仕様を満たすかを検証するが、図として得られている時間オートマトンを検証器の入力形式に変換しなければならない。時間オートマトンの検証後にクラス図の詳細化やテンプレートコードの生成を行うが、この際にもクラス図を詳細化プログラム・コード生成プログラムの入力形式に変換

する作業が必要となる。

ここで必要となる変換作業はソフトウェアの開発という観点からすると本質的な作業ではない。変換作業中に間違いが混入する可能性もある。そこでこの作業をツールによって支援する。つまりツール上で設計・作図を行い、直接時間オートマトンの検証、クラス図の詳細化、テンプレートコードの生成をできるようにする。このように変換作業をツールの内部で行い、別に用意されている検証プログラムをツールに取り込むことで開発者の作業量を削減する。

支援ツールを用いて開発を行う際の開発者の作業の流れを図 2.3に示す。図中の番号は図 2.2の中の番号と対応する。例えば (1) については、支援ツールを用いない場合は開発者自身が時間オートマトンの変換を行ない時間オートマトン検証ツールを起動しなければならないが、支援ツールを用いる場合はこれらの作業が支援ツールに検証を指示するだけに置き換わる。

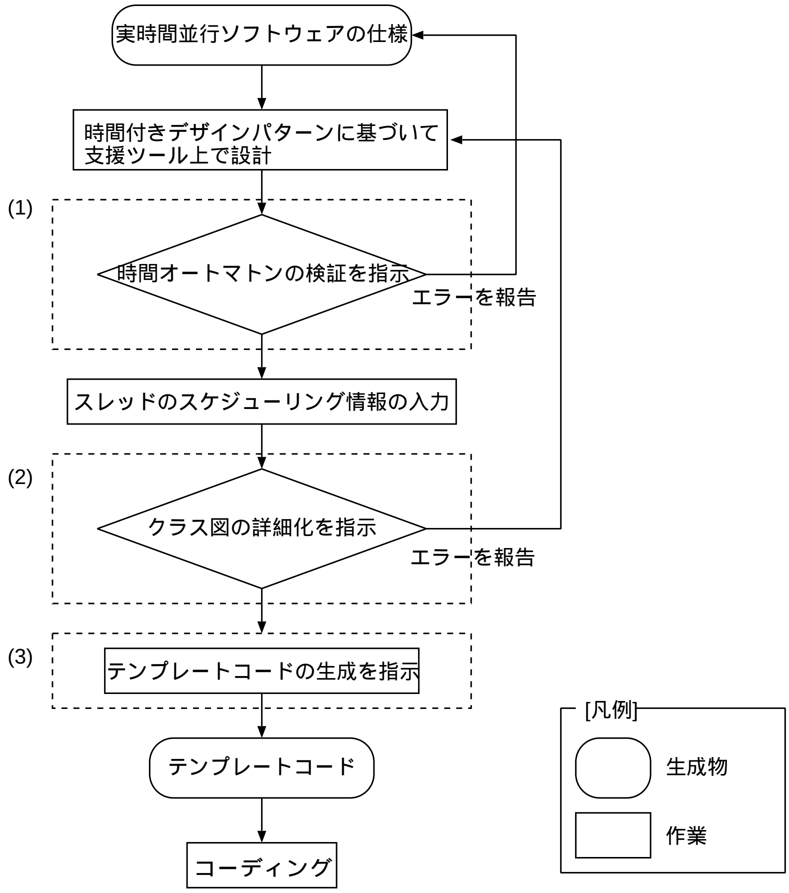

図 2.3: 支援ツールを用いた開発の流れ

第 3 章

開発支援ツール

3.1 機能

本研究で作成する開発支援ツールは以下に示す機能を持つ。

クラス図エディタ

UML によるクラス図をグラフィカルに編集する。クラスの名前や属性、あるいはコラボレーションの名前などノード要素に現れる文字列はダイアログなどを介することなく図中に直接記述することができる。また継承や関連など 2 つのノード要素の接続はその作成時に両端のノード要素の正当性を判定する。

時間オートマトンエディタ

時間オートマトンをグラフィカルに編集する。基本的な作図はマウスのみで行うことができる。状態や遷移それぞれの属性についてはダイアログから入力する。

時間オートマトンの検証

支援ツール上に描かれている時間オートマトンが与えられた仕様を満たすかを検証する。実際の検証には UPPAAL の verifyta というプログラムを用いる。図で表される時間オートマトンを verifyta の入力形式に変換し、

verifyta を起動する。そして verifyta の出力を受け取り、支援ツール上に表示する。

実装は Java を用いて行い、支援ツールの GUI には Swing コンポーネントを利用する。

3.2 システム構成

本システムのクラス構成は図 3.1のようになる。

図 3.1はシステムのモデルのみを表す。

図 3.1では UMLNode のサブクラスの一部と UMLLink のサブクラスを省略している。 UMLNode のサブクラスとしては図 3.1にある timed behavior ステレオタイプ化されたノートである TimedBehavior の他、クラスを表す UMLClass やコラボレーションを表す UMLCollaboration 等がある。また、 UMLLink のサブクラスとしては継承を表す UMLGeneralization や関連を表す UMLAssociation 等がある。これらは必要に応じて新たな要素を追加することも可能である。

クラスは大きく

- 基本クラス
- クラス図エディタ部
- 時間オートマトンエディタ部

に分けられる。

以下でそれぞれのクラスについて説明する。

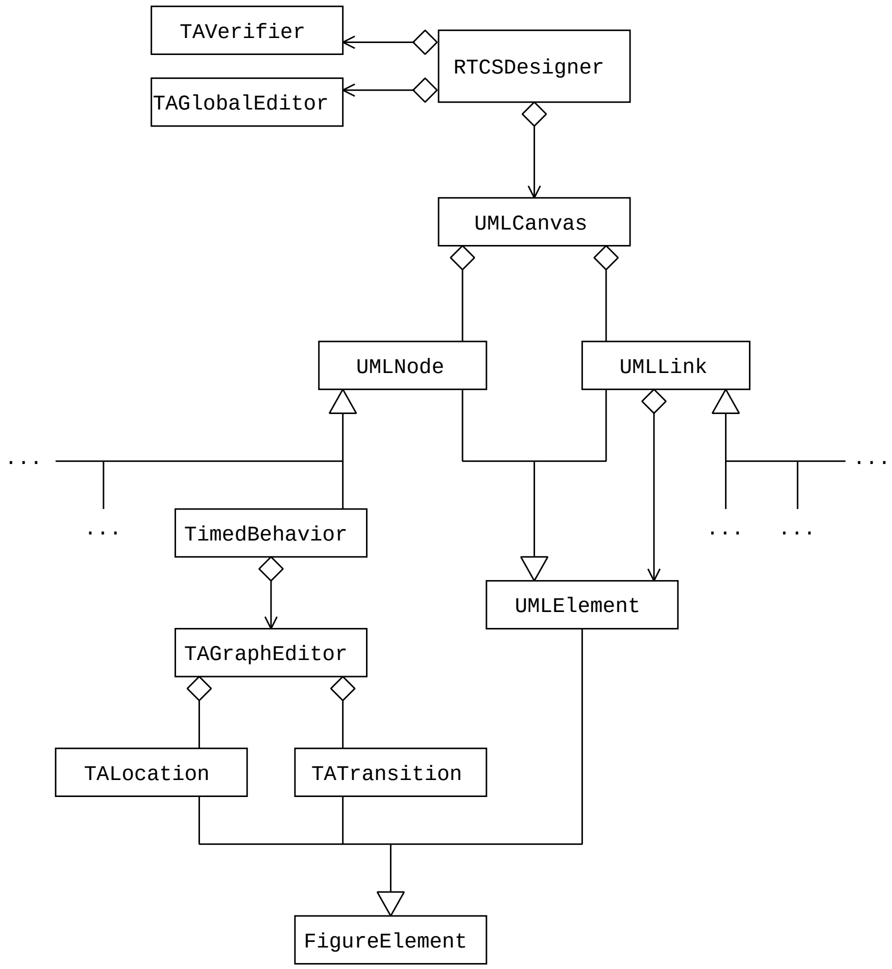

図 3.1: システムのモデルのクラス図（UML 表記）

3.2.1 基本クラス

RTCSDesigner

本研究で作成する支援ツールの本体である。ファイル操作やエディタクラス、時間オートマトン検証クラスなどの管理を行う。

時間オートマトンの検証はツール上に描かれている時間オートマトンと TAGlobalEditor のインスタンスに書かれている式を一時ファイルに出力して、TAVerifier のインスタンスに渡す。検証の際の TAVerifier の出力を表示するウィンドウを管理する。

FigureElement

クラス図や時間オートマトンに現れる図形要素のルートクラスである。

各図形要素はオーバーラップして描画されることもあるが、その時の上下関係を変更することができる。また各図形要素に対する操作のほとんどがポップアップメニューを通して行われるため、ポップアップメニューの管理を行う。

3.2.2 クラス図エディタ部

UMLCanvas

クラス図の編集を行う。

クラス図に現れる各要素の生成・削除など要素の管理を担当する。

UMLElement

UML のクラス図における図形要素のルートクラスである。

UMLNode

UML のクラス図をグラフとみなした場合のグラフのノードに相当する図形要素のスーパークラスである。このクラスのインスタンスは常に UMLLink クラスのインスタンスより上に描画される。

具体的な図形要素を表すサブクラスとして以下のようなクラスが存在する。

- **UMLClass**

  クラスを表す。名前、属性、操作といった要素を持つ。

- **UMLInterface**

  インターフェースを表す。実際には interface ステレオタイプ化されたクラスである。属性を持たないという点がクラスとは異なる。

- **UMLNote**

  ノートを表す。ノートはクラスや関連に付加されるコメントなどの情報を表示する。

- **TimedBehavior**

  timed behavior ステレオタイプ化されたノートを表す。時間制約を持つクラスの振舞いを記述する時間オートマトン及び同期イベントとメソッドの対応関係を表示する。

  TimedBehavior は UMLNote を継承する。

- **UMLCollaboration**

  コラボレーションを表す。コラボレーションとは、特定の目的において意味を持つ、一連のオブジェクトの集合である。

- **TimedDesignPattern**

  コラボレーションのうち時間制約を持つため時間オートマトンによって振

舞いが記述されるオブジェクト間のコラボレーションを表す。クラス図上では ≪timed behavior≫ 間のコラボレーションとして表示される。TimedDesignPattern は UMLCollaboration の特化であるため UMLCollaboration を継承する。

UMLLink

UML のクラス図をグラフとみなした場合のグラフのリンクに相当する図形要素のスーパークラスである。このクラスのインスタンスは常に UMLNode クラスのインスタンスより下に描画される。

具体的な図形要素を表すサブクラスとして以下のようなクラスが存在する。

・**UMLGeneralization**

クラスあるいはインターフェースの継承を表す。性質上リンクの両端は共にクラスまたは共にインターフェースでなければならない。

・**UMLRealization**

インターフェースの実装を表す。図形としては破線矢印で表されるが、矢印の先がインターフェースで反対側がクラスでなければならない。

・**UMLAssociation**

二項関連を表す。両端それぞれについて集約・コンポジション、ナビゲーション性、多重度を持つ。一端がノートになる、あるいは両端が timed behavior ステレオタイプ化されたノートになるのでなければ任意の 2 つの要素間の関連を表現できる。

・**UMLNoteAnchor**

ノートあるいは timed behavior ステレオタイプ化されたノートと他の図形要素を結びつけるリンクを表す。

TimedBehavior

timed behavior ステレオタイプ化されたノートである。時間オートマトンエディタを持つ。

3.2.3 時間オートマトンエディタ部

TAGraphEditor

時間オートマトンの編集を行う。

時間オートマトンの状態と状態間の遷移の管理を担当する。また時間オートマトン検証ツール verifyta の入力形式に沿って描かれている時間オートマトンの変換を行う。

TALocation

時間オートマトンの状態である。状態は以下の属性を持つ。

・**Name** 状態名を表す。

・**Invariant** この状態で存在できる条件であり、 false ならばこの状態には遷移できない。またこの状態の時に false になった場合は強制的に遷移する。

各状態は自身の関係する遷移の情報を保持する。それぞれの時間オートマトンにおいていずれか一つの状態が初期状態となる。

このクラスのインスタンスは常に TATransition クラスのインスタンスより上に描画される。

TATransition

時間オートマトンの遷移である。遷移は以下の属性を持つ。

・**Guard** 遷移が起こる時に満たされるべき条件を表す。

- **Synchronization** 遷移の際に送信あるいは受信するイベントである。
- **Method** イベントに対応するメソッドである。
- **Assignment** 遷移の際に発生するクロック変数への代入を表す。
- **Source** リンク元の状態を示す。
- **Destination** リンク先の状態を示す。

このクラスのインスタンスは常に TALocation クラスのインスタンスより下に描画される。

TAGlobalEditor

システム中の時間オートマトンすべてで共通に用いられるグローバル変数の宣言、時間オートマトンのインスタンス化式、システム内に現れる時間オートマトンのインスタンスの列挙、及び時間オートマトンの満たすべき仕様の以上 4 つを記述するエディタである。

TAVerifier

時間オートマトンの検証器である。時間オートマトンの検証系である UPPAAL に含まれる検証プログラム verifyta をラップする。 TAVerifier のインスタンスに verifyta のコマンドを渡し検証を指示すると、クラスの内部で verifyta を起動し検証を行う。また verifyta が出力する検証結果を処理する。

3.3 実装

ここではクラス図エディタのデータ構造と図の表示、及び時間オートマトン検証クラスの実装について述べる。時間オートマトンエディタはノードとリン

クがそれぞれ一種類ずつであるクラス図エディタとほぼ同様であると考えることができる。

3.3.1 クラス図エディタの構造

クラス図に表れる図形要素はクラス図エディタの中心的なクラスである UMLCanvas によって管理される。 UMLCanvas においては Java のライブラリで提供される可変長配列である`java.util.Vector` を用いて全要素を保持する。

以下 Java のライブラリで提供されるクラスについては`java` あるいは`javax` で始まる完全限定名を用いて表し、 Java のライブラリで提供されるとの断りはいれない。また UMLNode を継承する図形要素のことをノード、 UMLLink を継承する図形要素のことをリンクという。

ノードやリンクの実体は UMLCanvas が管理するが、それぞれのノードは自身の関係するリンクの参照、それぞれのリンクはその両端のノードあるいはリンクの参照を持つ。これは、例えばリンクはその種類によって無効な両端のノードやリンクが存在するため生成時にノードの有効性の判定を行うが、その時にリンク自身が両端のノードやリンクについて知っている必要があるからである。ノードやリンクの参照は UMLCanvas と同様に`java.util.Vector` を用いて保持する。

図形の表示は UMLCanvas で行うのではなく、`javax.swing.JLayeredPane` を継承する NodeLayer と LinkLayer に委譲する。 NodeLayer ではノード、 LinkLayer ではリンクの表示を行う。そのため NodeLayer はノードの参照を、 LinkLayer はリンクの参照を持つ。そして LinkLayer の上に NodeLayer を重ねることでノードとリンクがオーバーラップしても常にノードがリンクの上に描画される。

以上で述べているようにノードとリンクは相互に参照を持っている。そのた

めノードやリンクの削除は UMLCanvas から削除するだけでは不十分であり、以下の手順で行う。削除は削除対象の図形要素自身が行い、 NodeLayer あるいは LinkLayer からの削除は UMLCanvas から削除する時に同時に行う。

・ノードの削除

1. 自身の関係するリンクを削除する
2. UMLCanvas から自身を削除する

・リンクの削除

1. 両端のノードから自身を削除する
2. UMLCanvas から自身を削除する

ビューまで含めたクラス図エディタのクラス構成を図 3.2に示す。図 3.2では クラスによっては属性や操作の一部が記述されている。

個々の図形の描画は各図形自身が行う。これは一つの図形要素を一つの Java の Swing コンポーネントとして、コンポーネント自身を描画するメソッド `void javax.swing.JComponent#paintComponent(java.awt.Graphics)` をオーバーライドすることで実現する。

3.3.2 時間オートマトン検証クラス

時間オートマトンの検証ツールである verifyta をラップする。 verifyta の起動にはメソッド `java.lang.Process java.lang.Runtime#exec(java.lang.String[])` を用いる。そして起動したプロセスの出力をツールに表示する。基本的な動作は以上だが、次に述べる理由によりこのクラスは別のスレッドとして動作させる。

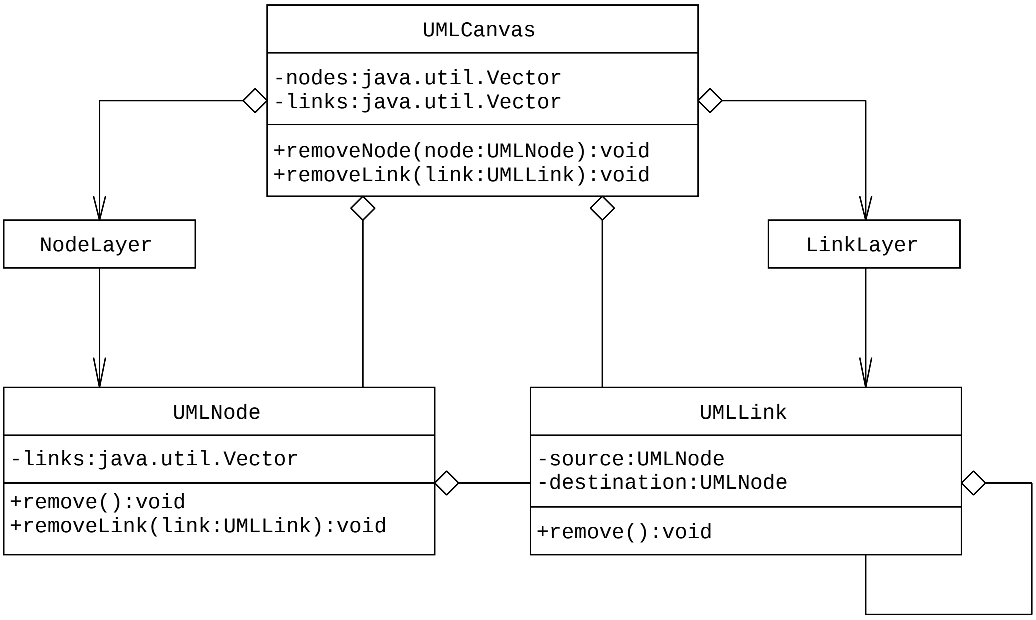

図 3.2: クラス図エディタのクラス図（UML 表記）

別のスレッドとせずにメインスレッドの中で動作させると、verifyta の出力を全て得るために verifyta の実行が終了するまでメインスレッドの実行を停止して待機する必要がある。この間マウス入力を初めとする全ての入出力動作がブロックされる。そのためユーザが verifyta の実行を途中で停止させることが不可能である。 verityta に異常が発生し終了しなかった場合にはブロックされたままになり何もできなくなる。

verifyta は複数の仕様について連続して検証が行うことが可能であり、一つの仕様について検証が終了するたびにその仕様が満たされるかどうか出力する。支援ツール上でも同様の動作をさせるため verifyta の出力を定期的に調べて表示したいが、そのためには別のスレッドとして動作させる必要がある。

第 4 章

例

本章では支援ツールを用いた開発作業の一例を示す。ただしソフトウェア全体の開発では例としては大き過ぎるので、簡単な例として TimedObserver パターン [2] を用いたマウスのイベントハンドラを対象とする。クラス図の詳細化やテンプレートコードの生成はそれぞれのプログラムが自動で行うので、ここでは開発者の作業の中心となる図 2.3の (1) の段階までの流れを示す。

4.1 問題と仕様

TimedObserver パターンを用いてマウスのイベントハンドラを設計する。ハンドリングするイベントは時間制約のあるイベントのうちクリックイベントのみとする。

クリックイベントの時間制約は、クリックイベントの間隔が一定値以下ならダブルクリックと判定し一定値より大きければシングルクリック 2 回と判定する、という制約である。

ここでは TimedObserver パターンの TimedObserver クラスを継承する MouseObserver、 TimedSubject クラスを継承する MouseSubject の各クラスを作る。

4.2 開発の流れ

4.2.1 起動

支援ツールを起動すると図 4.1のような画面が現れる。まず始めに時間オートマトン検証プログラム verifyta のパスを設定する。メニューを Automaton → Setting とたどると図 4.2に示すダイアログが現れるので、 verifyta のあるディレクトリを指定する。

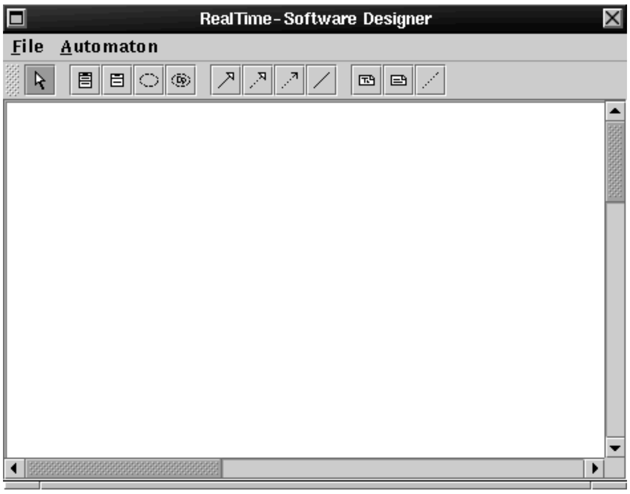

図 4.1: 支援ツールのメイン画面

4.2.2 クラス図の作成

クラス図の作成は次のようにして行う。

・ノード要素

1. 画面上部のツールバーで要素を選択する
2. 描画したい位置をクリックする

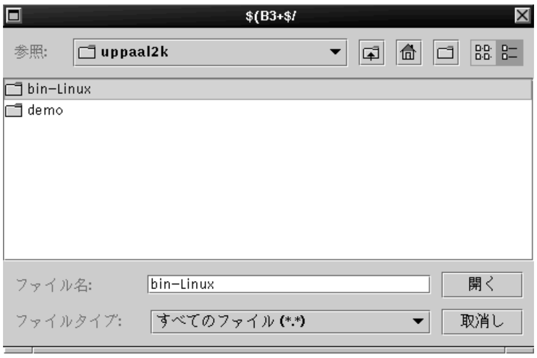

図 4.2: verifyta のパス指定画面

3. クラス名などの文字列の入力は入力したい領域をダブルクリックして行う (図 4.3)
4. 要素の移動はドラッグ、削除はポップアップメニューから行う

・リンク要素

1. 画面上部のツールバーで要素を選択する
2. リンク元となるノード要素を選択する
3. リンク先となるノード要素を選択する
4. ノード要素を選択する際不適な要素であれば警告される (図 4.4)
5. 削除などはポップアップメニューから行う

この例では以下のノード要素を配置する。

- TimedBehavior インターフェース
- TimedObject クラス
- TimedSubject クラス

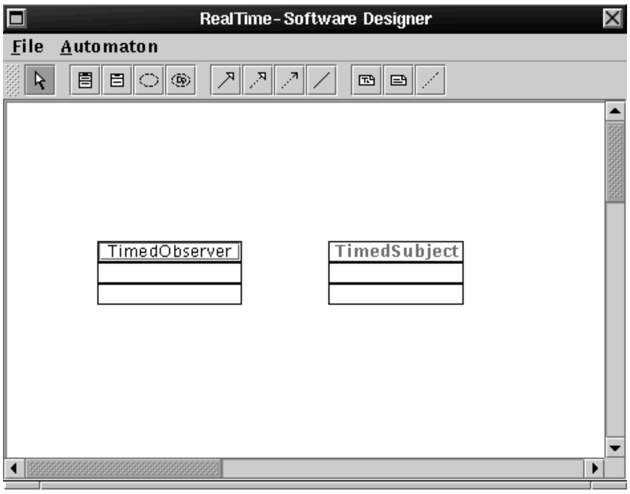

図 4.3: 文字列の入力

図 4.4: 不適なノード要素に対する警告

- MouseObject クラスと ≪timed behavior≫ ノート
- MouseSubject クラスと ≪timed behavior≫ ノート
- ≪timed design pattern≫ コラボレーション

これらの要素とそれぞれの間のリンクを作成したクラス図を図 4.5に示す。

4.2.3 時間オートマトンの作成

timed behavior ステレオタイプ化されたノートには時間オートマトンを記述するが、支援ツールでは次のようにして時間オートマトンの図を作成する。

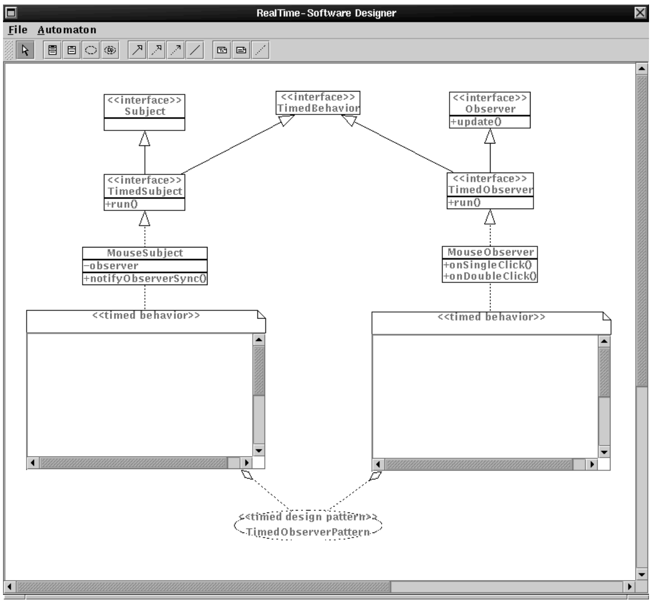

図 4.5: マウスイベントハンドラのクラス図

- 状態

1. 配置したい位置で左ダブルクリックする
2. 配置した要素のダブルクリックあるいはポップアップメニューから編集ダイアログ (図 4.6) を表示して編集する
3. 削除はポップアップメニューから行う

- 遷移

1. 時間オートマトンエディタの要素のない部分で右クリックし、ポップアップメニューから Transition Mode に移る。
2. 遷移元及び遷移先の状態を選択する
3. ダブルクリックあるいはポップアップメニューから編集ダイアログ (図 4.7) を表示して編集する

4. 削除はポップアップメニューから行う

時間オートマトンの名前の入力や内部で用いられる変数の宣言は、やはりポップアップメニューからダイアログ (図 4.8) を表示して行う。グローバル変数の宣言や時間オートマトンのインスタンスの列挙、時間オートマトンが満たす仕様の記述は、メニューを Automaton → Global Editor とたどり図 4.9のようなダイアログを開いて行う。

図 4.6: 状態の編集ダイアログ

図 4.7: 遷移の編集ダイアログ

図 4.5に時間オートマトンを記述したクラス図を図 4.10に示す。

4.2.4 時間オートマトンの検証

時間オートマトンの検証はメニューの Automaton → Verify で実行が開始される。実行が開始されるとウィンドウ (図 4.11) が一つ開き、 verifyta の出力が表

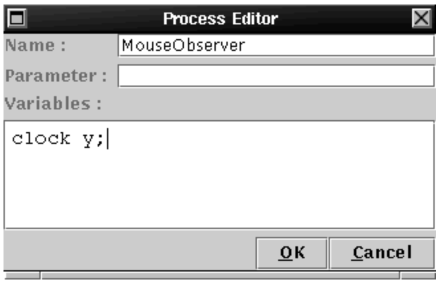

図 4.8: 名前入力・変数宣言ダイアログ

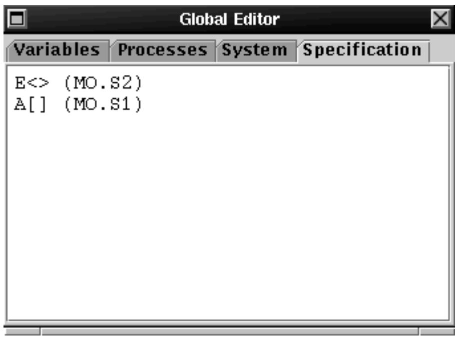

図 4.9: 仕様の記述

示される。

満たされる仕様に対しては、例えば `Property 1 (line 1) is satisfied.` など「satisfied.」、満たされない仕様に対しては `Property 1 (line 1) is NOT satisfied.` など「NOT satisfied.」と表示される。その他にもデッドロックの発生などを通知してくることもある。

検証が正常に終了した後は、スレッドのスケジューリング情報を入力してクラス図の詳細化を行い、テンプレートコードの生成へと進めていく。その後、生成されたテンプレートコードに対してコーディングを行っていく。

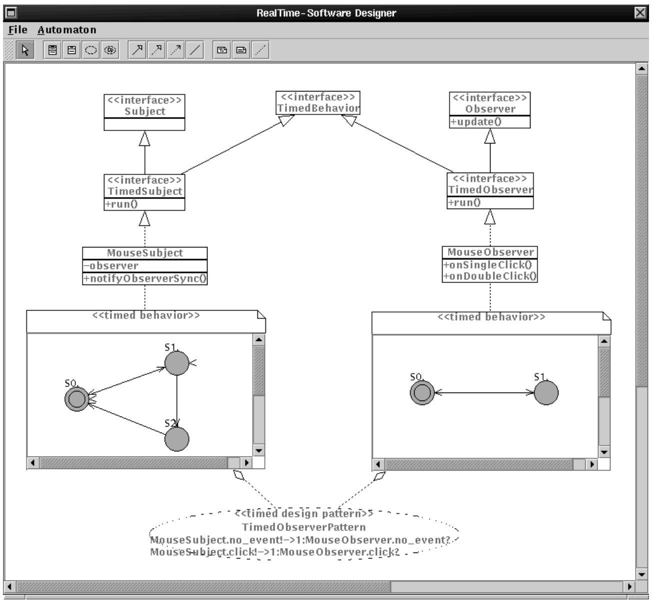

図 4.10: 時間オートマトンも記述したクラス図

図 4.11: 検証の出力表示ウィンドウ

第 5 章

おわりに

5.1　まとめ

本研究では、時間付きデザインパターンに基づく実時間並行ソフトウェア開発の支援ツールを提案し、GUI には Swing を用い Java によって実装した。

支援ツールの内部に時間オートマトン検証プログラム等の既存プログラム独自の入力データ形式を隠したため、開発者は既存プログラムのデータ形式について知る必要はなく、本来図であるクラス図や時間オートマトンを図のまま扱い開発を進めることが可能となった。

ツールによる支援を受けずに開発を進める場合は、時間オートマトンやクラス図を既存プログラムのデータ形式に変換するというソフトウェア開発の観点からは本質的ではない作業が必要となる。支援ツールを利用する場合は、ツールがこのような作業を開発者に代わって行うため変換作業中に誤りが混入する可能性がなくなる。それによって開発者はシステムの設計や分析といった作業に集中することができると共に、開発の省力化・効率化が期待できる。

開発は時間オートマトンの検証を行いながら進むため、ツールが生成するテンプレートコードは時間制約に関する部分は正しいことが保証される。コーディングはツールが生成したコードに追加する形で時間制約とは関係しない部分のみが必要であり、時間制約に関する部分は設計を行うだけでコーディングは不

要である。

5.2　今後の課題

今回作成した支援ツールは時間オートマトン検証プログラム、クラス図詳細化プログラム、テンプレートコード生成プログラムの 3 つの外部プログラムを使用する。しかし現時点で支援ツールに取り込まれているのは時間オートマトン検証プログラムのみであり、支援ツール上でテンプレートコードの生成まで行うことはできない。この点の実装を行い時間付きデザインパターンに基づく開発全体を支援できるようにする必要がある。

支援ツールではクラス図の基本的な要素は記述できるが、 UML で定義されているが記述できない要素もいくつか存在する。例えば、限定子やパッケージ、パラメタライズドクラスなどである。これらの記述も行えることが望ましい。支援ツール側で時間付きデザインパターンのテンプレートを用意できれば開発者の作業量をさらに削減できるが、このためには時間付きデザインパターンについてさらに研究を進める必要がある。

謝辞

本研究を進めるにあたり、熱心に御指導頂いた阿草清滋教授、結縁祥治助教授に感謝致します。また、熱心に議論して頂いた愛知県立大学山本晋一郎助教授、名古屋大学濱口毅助手及び阿草研究室の皆さん、特に時間付きデザインパターンについて数々の助言を頂いた高木豊典氏に感謝致します。

参考文献

[1] 高木豊典, 結縁祥治, 阿草清滋, ”オブジェクト指向プログラミングにおける時間付きデザインパターン ”, 日本ソフトウェア科学会第 17 回全国大会,2000

[2] 高木豊典, 結縁祥治, 阿草清滋, ”時間付きデザインパターンに基づく実時間並行ソフトウェアの開発手法 ”, 情報処理学会第 31 回プログラミング研究会,2000

[3] Erich Gamma, Richard Heml, Ralph Johnson, John Vlissides, ”Elements of Reusable Object-Oriented Software, Design Patterns”, ADDISON-WESLEY, 1994

[4] Paul Pettersson and Kim G. Larsen, ”UPPAAL2k”, Bulletin of the European Association for Theoretical Computer Science, volume 70, pages 40-44, 2000

[5] Kim G. Larsen, Paul Pettersson, and Wang Yi, ”UPPAAL in a Nutshell”, International Journal on Software Tools for Technology Transfer, pages 134-152, 1998

[6] Rational Software Corporation, ”UML(Unified Modeling Language) 表記法ガイド ver.1.1”, 1997

付録 A

マニュアル

A.1　概要

本システムは時間付きデザインパターンに基づく実時間並行ソフトウェア開発を支援する．本システムが提供するクラス図及び時間オートマトンエディタにより実時間並行ソフトウェアの設計をグラフィカルに行うことが可能である．また，時間オートマトンの検証系である UPPAAL と連携することで時間オートマトンの検証を本システム上から行うことができる．

A.2　インストール

A.2.1　前提

本システムは Java を用いて実装されているため，使用には Java の利用できる環境が必要である． Java のバージョンは Java2 すなわち 1.2.x 及び 1.3.0 を推奨する．なお， Java のバージョンが 1.1.x でも使用することはできるが， Swing コンポーネントを使用しているため，その際には別途 JFC/Swing1.1.1 が必要となる．

時間オートマトンの検証に UPPAAL を利用するため，`http://www.uppaal.com/` からダウンロードしてインストールしておく必要がある．

A.2.2 インストール

Java のバージョンが 1.2.x 及び 1.3.0 以上の場合

1. rtdesigner.jar を適当なディレクトリに置く

Java のバージョンが 1.1.x の場合

1. rtdesigner.jar を適当なディレクトリに置く
2. コマンドラインから `jar -xvf rtdesigner.jar` と入力し解凍する

A.3 使用方法

A.3.1 起動

`rtdesigner.jar` をおいたディレクトリのパスを `RTD_HOME` とする．

Java のバージョンが 1.2.x 及び 1.3.0 以上の場合

1. コマンドラインから `java -jar RTD_HOME/rtdesigner.jar` と入力する

Java のバージョンが 1.1.x の場合

1. `RTD_HOME` に移動する
2. コマンドラインから `java designer.RTCSDesigner` と入力する

A.3.2 基本操作

クラス図の作成

クラス図の作成は次のようにして行う．

・ノード要素

1. 画面上部のツールバーで要素を選択する
2. 描画したい位置をクリックする
3. クラス名などの文字列の入力は入力したい領域をダブルクリックして行う
4. 要素の移動はドラッグ，削除はポップアップメニューから行う

・リンク要素

1. 画面上部のツールバーで要素を選択する
2. リンク元となるノード要素を選択する
3. リンク先となるノード要素を選択する
4. ノード要素を選択する際不適な要素であれば警告される
5. 削除などはポップアップメニューから行う

クラス図のポップアップメニューは表 A.1の中から要素と状況に依存して必要なものが表示される．

時間オートマトンの作成

時間オートマトンはクラス図の timed behavior に記述する．次のようにして作成する．

・状態

1. 配置したい位置で左ダブルクリックする
2. 配置した要素のダブルクリックあるいはポップアップメニューから編集ダイアログを表示して編集する

Delete XXX To Front To Back	その要素を削除 最前面に移動 最後尾に移動
Add Attribute Delete Attribute Display Attributes Hide Attributes	クラスの属性の追加 クラスの属性の削除 クラスの属性を表示 クラスの属性を非表示
Add Operation Delete Operation Display Operations Hide Operations	クラスの操作の追加 クラスの操作の削除 クラスの操作を表示 クラスの操作を非表示
Set Stereotype Delete Stereotype Display Stereotype Hide Stereotype	クラスのステレオタイプを設定 クラスのステレオタイプの削除 クラスのステレオタイプを表示 クラスのステレオタイプを非表示
Add Event Delete Event Display Events Hide Events	≪timed design pattern≫ のイベントの追加 ≪timed design pattern≫ のイベントの削除 ≪timed design pattern≫ のイベントを表示 ≪timed design pattern≫ のイベントを非表示
Rollover Expand Fold	true の時マウスが離れると自動的に折り畳む 折り畳まれたノートを開く ノートを折り畳む

表 A.1: クラス図のポップアップメニュー

3. 削除はポップアップメニューから行う

・遷移

1. 時間オートマトンエディタの要素のない部分で右クリックし，ポップアップメニューから Transition Mode に移る．
2. 遷移元及び遷移先の状態を選択する

3. ダブルクリックあるいはポップアップメニューから編集ダイアログを表示して編集する
4. 削除はポップアップメニューから行う

A.3.3 ツールバー

ツールバーには配置するクラス図の要素が列挙されている．左から以下の要素である．

- **Select** 配置されている図形要素を選択する
- **Class** クラスを配置する
- **Interface** インターフェースを配置する
- **Collaboration** コラボレーションを配置する
- **TimedDesignPattern** ≪timed behavior≫ 間のコラボレーションを配置する
- **Generalization** 汎化を示す
- **Realization** インターフェースの実装を示す
- **Dependency** 依存性を示す
- **Association** 集約・コンポジション・ナビゲーション性を含む関連を示す
- **TimedBehavior** 時間オートマトンを記述できるノートを配置する
- **Note** 通常のノートを配置する
- **NoteAnchor** ノートと他の要素を連結する

A.3.4 メニュー

- File
  - **New** 新しいクラス図を作成する
  - **Open** 保存されているクラス図を開く
  - **Save** 描かれているクラス図を保存する
  - **Exit** 終了する
- Automaton
  - **Verify** 時間オートマトンを検証する
  - **Global Editor** グローバル変数や使用を記述するエディタの表示 / 非表示を切替える
  - **Setting** 検証器 verifyta のパスを指定する

付録 B

ソース