実世界指向自己適応フレームワークにおける動的検証メカニズム関する調査研究

代表研究者　　中　川　博　之　　大阪大学　大学院情報科学研究科　准教授

1 はじめに

近年のソフトウェアシステムでは，設計時に実行時の環境を完全に想定することが難しく，システム実行時に動的に振る舞いを変更させることのできる自己適応システムが，次世代ソフトウェアシステムのあるべき設計スタイルとして注目を集めている．自己適応システム[1][2]の活躍が期待される場は，無人車両（UMV）や無人航空機（UAV）などの技術発展に伴って，組み込みシステムの分野へも広がりを見せている．

安全面などの理由から，組み込みシステムの物理的な動作に関わる時間制約は重要視されることがあり，時間制約に基づいた適応動作は組み込みシステムの自己適応化において重要な要件となり得る．その実現には動的な時間制約の検証が必須となるが，時間を扱うことのできるフレームワークは未だ確立されていない．本調査研究では，実世界を対象とする組み込み機器上で動作可能な自己適応ソフトウェアのプログラミングフレームワークを構築し，実際に組み込み機器上で動作させることで，その有効性を評価することを目的とする．その特徴は，（１）適応後の振舞いが正しいことを時間的制約内で保証するための動的検証メカニズムを備えている点，（２）組込みシステムのような時間的・空間的な制約をもつ環境下においても，これらの特徴を備えた十分に軽量化されたプログラミングフレームワークを実現する点にある．

2 軽量フレームワークの設計と実装

本調査研究では，まず組み込みシステムのような時間的・空間的な制約をもつ環境下においても動作可能な軽量化された自己適応システムプログラミングフレームワークを設計，実装した．このフレームワークは，研究代表者の先行研究である自己適応システム実装フレームワーク[3][4]を基盤としている．以降，先行研究の自己適応システム実装フレームワークを基本フレームワーク，新たに実装したフレームワークを軽量フレームワークと称する．

基本フレームワークは，自己適応システムが備えるべき適応機能を実装するために，並列する複数プロセスをエージェントが並列実行するタスクと見立てて実装するというアプローチに基づいている．基本フレームワークはこのアプローチの実現に，Java 言語を用いて実装されたエージェントプラットフォーム JADE [5]を拡張利用している．基本フレームワークでは，JADE が提供する Behaviour クラスを拡張した ComponentBehaviour クラスを導入することで，Component が提供すべき各機能を同クラスに実装し，複数のプロセスにより並行動作されるように拡張している．しかしながら，基本フレームワークは，JADE プラットフォームが動作するサーバ，PC 環境での動作を前提としたものであり，JADE には GUI 依存の操作や動作要件の制約から，使用する実世界ハードウェアによっては正常に動作できない場合があり，軽量フレームワークとしての要件を満たしているとは言えない．例えば，本研究にて予備実験で用いた Mindstorms [6]上においても上記理由により，基本フレームワークをそのまま動作させることはできない．そこで，まずは，基本フレームワークにて用いられている構造や API はそのままに，JADE に依存している機能を別途実装することで，Mindstorms 等の軽量プラットフォームにおいても動作可能な軽量フレームワークを設計，実装した．

図 1 は基本フレームワークと実装フレームワークのクラス関係図である．基本フレームワークにおける Agent，Behaviour，SimpleBehaviour の 3 つのクラスは JADE が提供するクラスである．JADE を利用することなく基本フレームワークの動作を再現するためには，これらの 3 つのクラスを Java が提供するクラスやその拡張クラスを用意することで代替する必要がある．本研究では，Java が提供する Thread クラスを利用す

ることとした. Thread クラスは複数の並列タスクを記述するためのクラスであり，JADE も Behaviour クラスの実装に同クラスを利用している．Thread クラスを拡張することで，基本フレームワークにおける ComponentBehaviour クラスの代替となる Component クラスを導入する．また，基本フレームワークにおける SelfAdaptiveAgent クラスに相当する ThreadManager クラスを実装し，並列動作させるべきコンポーネントの動作を管理・制御する機能を持たせた．ThreadManager クラスは，各時点で並行に動作されるべきコンポーネントのリストを持ち，start メソッドが実行されると，各コンポーネントに対応するスレッドが並行に処理を開始する．スレッドが実行されたコンポーネントは，自身の run メソッドが自動的に実行され，同メソッドにより抽象メソッドである action メソッドが実行される. そして, 各コンポーネントにおいて action メソッド内で後述の mode 変数の遷移条件を記述し，コンポーネントとして期待する動作内容を perform メソッドとして記述する．このように，クラス・メソッドを定義することにより JADE を利用しない軽量フレームワークにおいても，Component クラスを継承して各コンポーネントを記述することで並行動作が実現される．

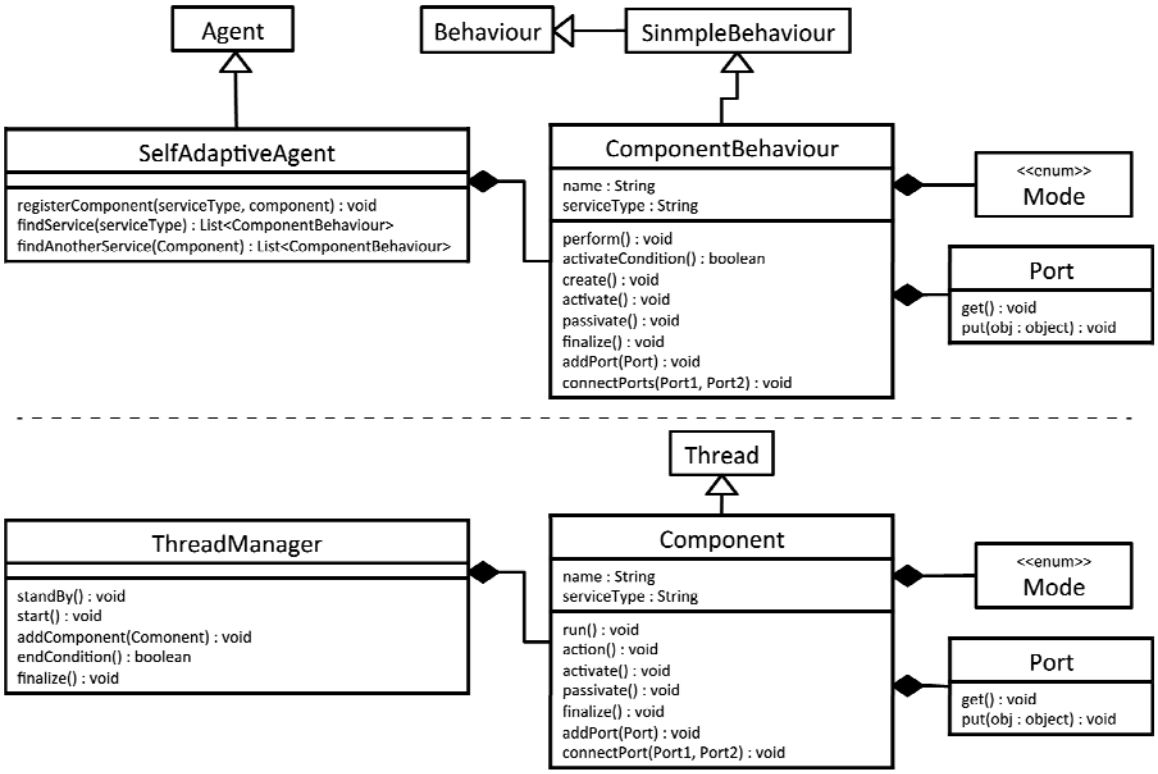

図 1 フレームワークの主要クラスと主要メソッド. 上段:基本フレームワーク, 下段:軽量フレームワーク. 基本フレームワークにおける Agent, Behaviour, SimpleBehaviour クラスは JADE が提供するクラスである.

3 時間制約を考慮可能な動的検証メカニズムの設計と実装

自己適応システムは，システムを取り巻く環境や内部状態などの情報を監視，分析した結果に応じて自らの振る舞いを刻々と変化させるメカニズムを備えたシステムであり，無人車両，無人航空機等の技術発展に伴い，組み込みシステムの分野においてもその活躍が期待されている．組み込みシステムは物理的な動作を伴う場合も少なくなく，機能要求や安全設計においてシステム処理に厳格な時間制約が設けられることがある．組み込みシステムの自己適応化によって，システム実行時に起こった時間制約違反に対して振る舞いを変更することで，再び制約が満たされるような状態に遷移することが期待される．このためにはシステム実

行中における動的な時間制約の検証が必要であり，動的検証の枠組みの提案[7][8][9] も行なわれている．しかし，自己適応システムの実装を目的としたフレームワーク[4][10][11] が様々な観点から提案されているが，時間を扱うことのできるフレームワークは未だ確立されていない．

本研究では，時間制約を記述することのできる動的検証メカニズムを設計し，プロトタイプを実装した．この動的検証メカニズムは時間制約の動的検証を用いた自己適応システムを実現させるための設計手法と，実装をサポートする Java 言語によるメソッドによって構成される．この設計手法においては，検証用の状態遷移モデルを事前に作成し,実行時に用いる.システム実行時にシステムの時間状態遷移モデルを設計し，フレームワーク内でそのモデルを保持している．図 2 は時間状態遷移モデルのクラス間関係を示したものである．時間状態遷移モデルを保持することにより，システム実行中にシステム内部情報やシステム外部環境の変化をモデルに反映させることができる．これにより，保持している時間状態遷移モデルに対してモデル検査を実行することで，システム実行中における動的な時間制約の充足判定を可能にしている．時間制約の充足判定に時間拡張された状態遷移図をモデルとするモデル検査ツールである UPPAAL の検証器を利用している．この検証器を用いることで，システム実行中に起こる時間制約違反を検出し，適応動作の実行を実現させる．システムの時間制約違反が検出されると，それらを解決することのできるシステム構成を検索し，構成変更により解決が可能である場合には新たなシステム構成による状態遷移モデルを提示する．つまり，システムの状態変化に対する適応手段が状態遷移モデルとして示される．本調査研究ではコンポーネントの接続によるシステム構成を想定している．そこで，システムを構成するコンポーネントの持つ責務を状態遷移図におけるロケーションとすることで，状態遷移モデルの変更をコンポーネントの組み替えとして読み取ることができる．したがって，示された状態遷移モデル通りにコンポーネントを組み替え，システムの構成を変化させることでシステムの内部情報や外部環境の変化に適応することができる．

3-1 UPPAAL

UPPAAL［12］は時間制約を扱うことのできるモデル検査ツールであり，時間制約を記述できるよう拡張された状態遷移図をモデルとして扱う．UPPAAL にはクロック型の変数が提供されており，全てのクロック変数は同じタイミングでインクリメントされ，個別にリセットすることができる．この変数を用いることで，状態遷移のタイミング制限や時間制約の条件を記述することができる．UPPAAL は Java で記述されたグラフィカルエディタと C++によって記述された検証器を組み合わせた構造となっており，グラフィカルエディタを用いて作成した時間状態遷移モデルと時相論理を用いた検証式を検証器へ入力として与えることでモデル検査を実行する．

3-2 状態遷移モデルの作成

提案手法では，UPPAAL のグラフィカルエディタを用いて状態遷移モデルを作成し，提案フレームワークには同エディタで作成した状態遷移モデルを XML 出力したものを与えている．状態遷移モデルは，時間制約の記述を容易にするために，システムの持つ機能単位でひとつのループとなる構成で設計する．各機能は抽象度の高い表現をした処理の連結で構成し，各処理の詳細を別テンプレートにて同様の処理連結によって表現する．この際，段階的に具体的なコンポーネントによる処理へと詳細化していき，最終的に単一のコンポーネントで実行可能な単位となるまで分解する(図 3)．この手順によって出来上がったテンプレート群が，フレームワークに対して初期入力として与える状態遷移モデルとなる．また，分解元のロケーションと分解先のテンプレートが担う処理をタスクとして，ある処理を別テンプレートに分解することをタスク分解としてそれぞれ定義する．さらに，タスク分解によって生成されたテンプレートのうち単一のコンポーネントで完遂できるタスクであるものモジュール，そうでないものをストラテジーと定義する．このタスク分解によるテンプレート生成の際に，対象システムの仕様に従って初期構成に含まれていないコンポーネントに対応するテンプレートや，既に組み込まれているものの代替となるストラテジーに対応するテンプレートも同様に作成しておく．これによって，コンポーネントの代替による時間制約違反への適応が可能となる，モジュール内の各処理において，その処理時間を定義することで，分解元のストラテジーや機能の実行に必要な時間が計算できる．ここで定義する処理時間は初期設定であり，システム起動直後や構成変更により該当コンポーネントが新たに利用されるようになった際に用いられる値である．システム実行中においては，動的な検証を行うために実際の計測値や他の情報を用いた類推値で適宜処理時間を更新する必要がある．この情報を

基に時間制約の充足判定が行なわれ，制約が満たされていなければ，解決することができるシステム構成の検索が実行される．

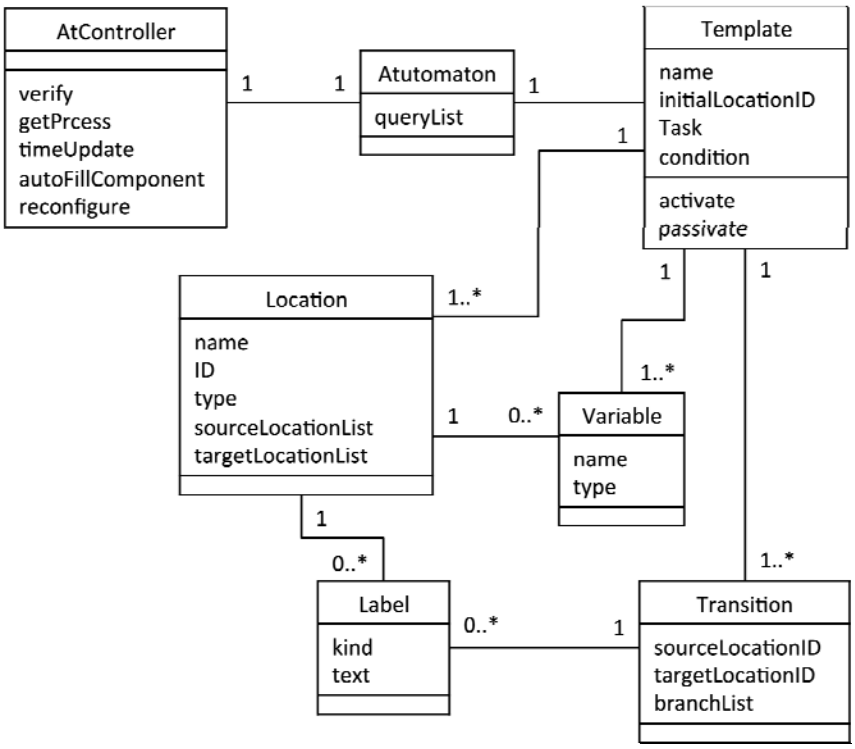

図２　状態遷移モデルのクラス間関係

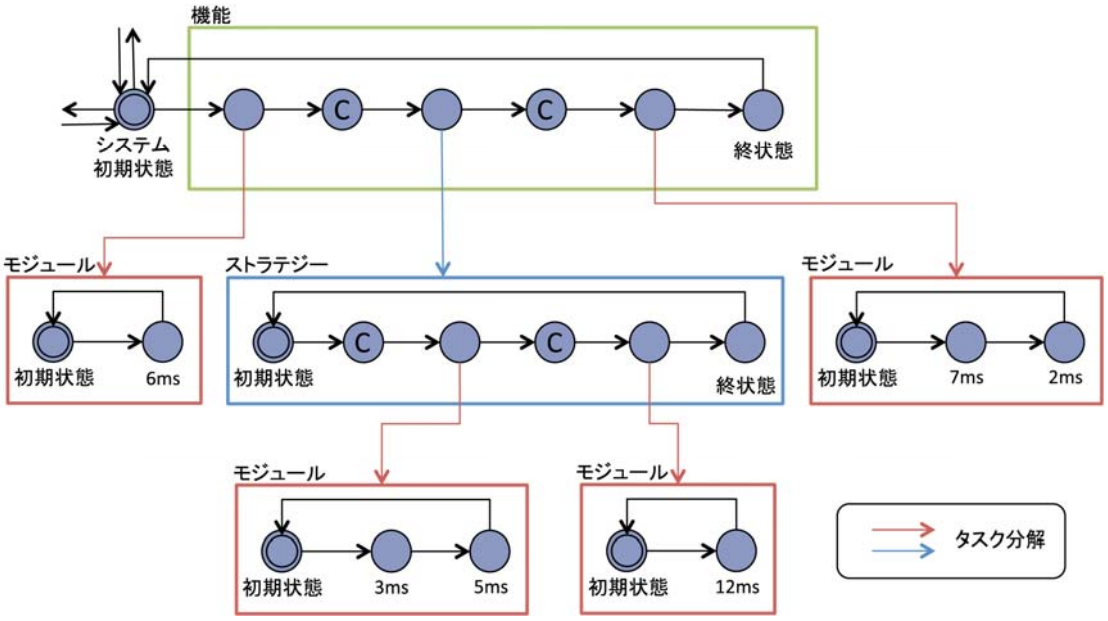

図３　機能のタスク分解

各モジュールに処理時間を定義し，時間検証には UPPAAL の同期通信機能を用いて複数のテンプレート間で同期をとり，遷移のタイミングを制御する．図 4 のように分解元のロケーションへ遷移するトランジショ

ンと分解先テンプレートの初期状態から遷移するトランジションで同期をとり，同様に分解先テンプレートの初期状態へと遷移するトランジションと分解元のロケーションから遷移するトランジションで同期をとる．これにより，それぞれのテンプレートにおける遷移のタイミングは図 4 右側のシーケンス図のようになる．

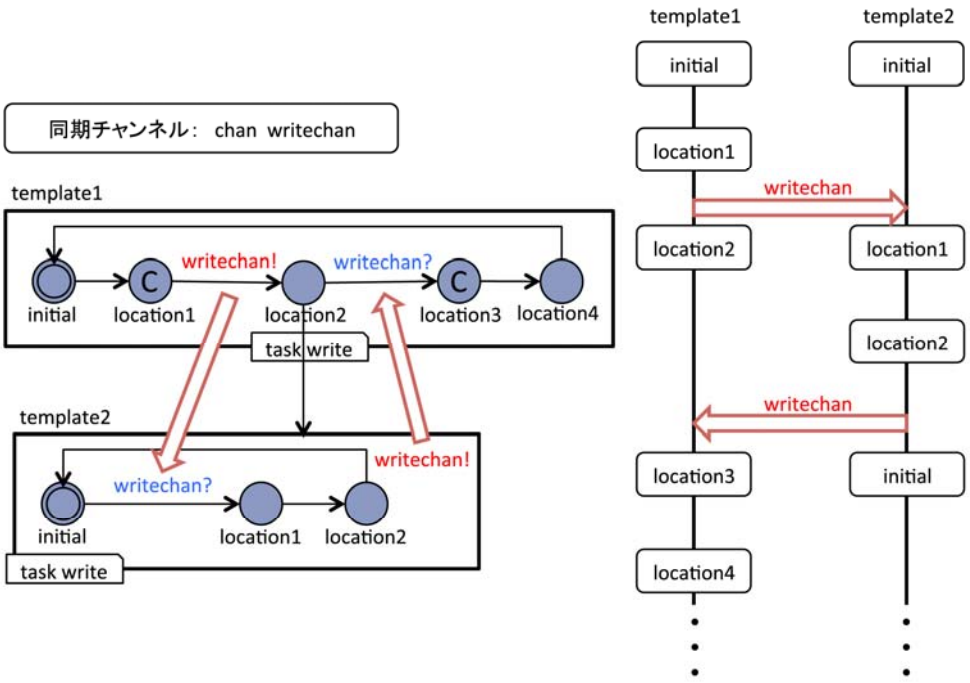

図４　チャネルを利用した同期メカニズム

時間制約は時相論理式で記述する．時相論理演算子は UPPAAL で用いられている以下のものを用いる．

- A[] p: 全ての実行パスで常に p が成り立つ
- A<> p: 全ての実行パスでいずれ p が成り立つ
- E[] p: p が常に成り立つパスがある
- E<> p: いずれ p が成り立つパスがある
- p --> q: p が成り立てばそのうち q が成り立つ

述語 p と q にはロケーションへの到達の他に，クロック変数を用いた評価を記述することができる．状態遷移モデルを独立した機能単位のループによって構成したことにより，初期ロケーションにてクロック変数が初期化されるよう記述することで，各機能の終了を表す最終ロケーションへの到達とクロック変数の値を使った時間条件を論理式にて表現し，機能に課すべき時間制約を表現することができる．

4 実験

本調査研究では,時間制約を考慮可能な動的検証メカニズムに対して,シミュレーション実験を実施した．この実験では小型無人航空機の制御システムを題材とし，3 節で述べた設計手順に従って検証メカニズムを実験的に利用する．また，シミュレーションシナリオに基づき時間制約が満たされなくなった際のシステムの振る舞いを確認する．

4-1 例題システム

この実験では小型無人航空機の制御システムを題材とする．このシステムは，用意されたコマンド群のうち指定されたものを実行できるよう内部状態やセンサー類，駆動パーツなどを制御することを目的としたシステムである．システムが実行すべきコマンドは，操縦者から無線通信にて逐次送信されると想定する．コマンドを同時に複数実行することはできず，コマンド実行中に別のコマンドが送信された場合キューに登録されずに破棄される．表 1 はシステムに用意されたコマンドの一部を示したものである．

このシステムは，地上にて静止している待機状態と，地上を離れ空中を飛行している飛行状態の 2 種類の状態を持つ．また，機体には高度センサー，加速度センサーが搭載されており，個別のモーターに接続された 4 つのプロペラの回転数を制御することにより飛行する．バッテリー枯渇による墜落を防止するために，バッテリー残量が 20 %を下回ると強制着陸させる．システムに課す時間制約の条件として，全ての機能に対して時間制約を設けることとする．ただし，安全の観点から命令解読，水平移動，垂直移動，回転，姿勢制御の機能に関しては許容される遅延を短くし，制約を厳しいものとする．その他の機能についてはある程度の遅延を許容できるような制約を課すものとする．

4-2 システム設計

3 章で述べたように機能単位でのループを作るため，システムに持たせる機能から設計する．本実験ではシステムに持たせるべき機能として以下の 10 の機能を用意した．

1. 命令解読（CommandDecode）：受信したコマンドを解読し，機能の実行系列を決定する．
2. 離陸（TakeOff）：待機状態ならば一定の高さまで浮上し，飛行状態へと移行させる．
3. 着陸（Landing）：飛行状態ならばその場で降下し，プロペラを停止させる．停止後に待機状態へと移行させる．
4. 姿勢制御（BalanceControle）：加速度センサーから情報を取得し，分析する．分析結果に従ってその場で静止できるような各モーターの回転数算出し，その値に従ってモーターを回転させることにより機体を安定させる．
5. バッテリー確認（CheckBattery）：バッテリー残量を確認し，強制着陸実行のための設定値と比較する．規定値を下回り，強制着陸が必要と判断された場合は即座に実行する．
6. 充電（ChargeBattery）：待機状態であればバッテリーへの充電を行う．
7. 水平移動（MoveHorizontal）：指定された方向へと水平に機体を移動させるために必要な各モーターの回転数を算出し，その値に従ってモーターを回転させる．
8. 垂直移動（MoveVertical）：指定された方向へと垂直に機体を移動させるために必要な各モーターの回転数を算出し，その値に従ってモーターを回転させる．
9. 回転（Turn）：指定された方向へと機体を回転させるために必要な各モーターの回転数を算出し，その値に従ってモーターを回転させる．
10. 宙返り（LoopTheLoop）：宙返り実行に十分な高度があるかを高度センサーにより確認し，十分な高度があると判断されればモーターを回転させ，宙返りを実行する．

各コマンドについてこれらの機能を用いて実現させるための実行機能系列を定義する．表 1 内に各コマンドの実行機能系列は機能に添えられた番号を用いて示している．

表 1　コマンド一覧（一部掲載）

コマンド	待機状態	飛行状態	実行機能系列
離陸	○	×	1→2→4→5
着陸	×	○	1→3→5
充電	○	×	1→6
前進	×	○	1→7→4→5
後退	×	○	1→7→4→5
...	...	...	...

次に，定義した各機能について処理系列を作り，状態遷移図を作成する．作成した状態遷移図内の各処理について 3 節で述べたタスク分解を行い，単一コンポーネントによる処理まで詳細化する．例えば，離陸機能は状態確認，上昇，状態変更の 3 つの処理からなり，このうち状態確認，状態変更は共に単一コンポーネントによる処理が可能であるため，それぞれ状態モジュール，状態コントローラーモジュールへとタスク分解を実行する．残る上昇処理には初期化，プロペラ回転の 2 工程が必要だが，これらは単一コンポーネントでは処理できないため，この 2 つの処理を持つ上昇ストラテジーとして分解する．上昇ストラテジーが持つ 2 つの処理である初期化とプロペラ回転は共に単一コンポーネントで処理できるため，それぞれイニシャライザモジュール，モーターモジュールへとタスク分解を実行する．この行程を繰り返し，完成した全ての機能の状態遷移図を統合したものが図 5 であり，図中の赤い数字は機能番号と対応している．さらに，初期構成に含まれていないいくつかのコンポーネントについても対応するモジュールを作成しておく．最終的に，このシステムでは図 5 に示した統合された機能状態遷移図に加え，4 つのストラテジー，17 のモジュールの合計 22 の状態遷移図が作成された．また，タスク分解により作成された各モジュール内の処理について，処理時間の初期値を定義しておく．ここまでの行程でシステムが持つ全ての状態遷移図が完成したため，定義した処理時間の初期値を添えて初期構成に必要な各テンプレートをインスタンス化し，システム構成として登録する．

最後に，システムが満たすべき時間制約を定義する．本シミュレーション実験ではシステムの持つ各機能に対して時間制約を設ける．表 2 は各機能に設けた時間制約一覧である．各機能について，処理系列に含まれるタスクに定義した処理時間から求めた理論値と設定された時間制約を比較し，許容される遅延を計算している．許容遅延が赤文字で記されている機能については，安全面などの理由から大きな遅延が許されず，時間制約が厳しくなっている機能である．表内の各項目について，クロック変数と対応するロケーションを用いた時相論理式で記述し，クエリとして登録する．

表 2　時間制約一覧

機能	時間制約	理論値	許容遅延
1.　命令解読	10ms	8ms	2ms
2.　離陸	40ms	29ms	11ms
3.　着陸	40ms	27ms	13ms
4.　姿勢制御	35ms	33ms	2ms
5.　バッテリー確認	10ms	3ms	7ms
6.　充電	115ms	103ms	12ms
7.　水平移動	40ms	37ms	3ms
8.　垂直移動	43ms	40ms	3ms
9.　回転	40ms	37ms	3ms
10.　宙返り	30ms	23ms	7ms

4-3 シミュレーションシナリオ

前節にて設計したシステムを用いてシミュレーション実験を行った．実験のために下記の 2 つのシナリオを用意した．これらのシナリオを用いたシミュレーションにより，時間制約違反が発生した場合における，システムの振る舞いを確認した．本調査研究では時間制約違反が発生する要因としてコンポーネント故障を想定したシミュレーションを行っている．故障としては，ハードウェアにおける物理的なものとバグなどによるソフトウェアに起因するものを扱った．提案メカニズムが時間制約の動的検証によって制約違反を検出し，適応動作が実行されたものであることを示し，時間制約の動的検証により振る舞いの変更が可能な自己適応システムの実現に有効であることを明らかにする．

シナリオ 1．状態確認コンポーネントの故障：機体には地上にて待機中，あるいは飛行中であるかを表す

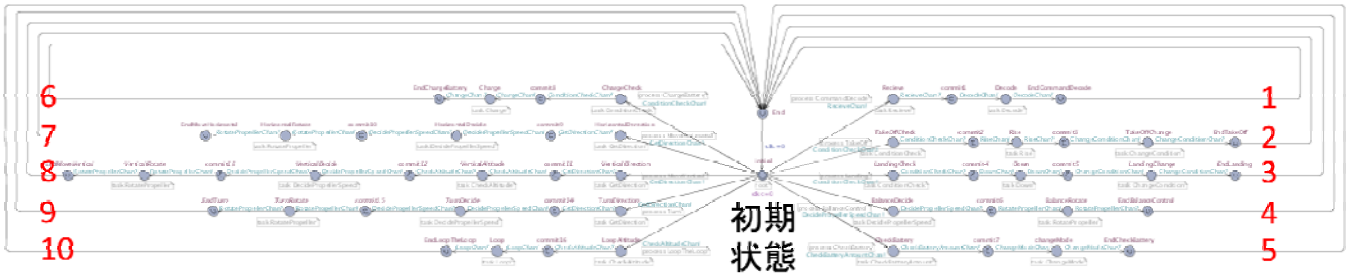

図 5　小型無人航空機制御システムの最上層状態遷移図

現在の状態が記憶されており，それを確認するためのコンポーネントに対応するモジュール“ CondChecker ”が存在する．このシナリオではこのコンポーネントが故障してしまったことにより現在の状態を取得できなくなってしまい，このコンポーネントを用いている実行機能系列を最後まで実行することができず，結果的に処理時間が大幅に増加したと見なされ時間制約を満たすことができなくなってしまった場合を想定する．

実行系列の中断が制約違反として検出されると，モジュール“ CondChecker と同じ状態確認タスクを持つような別のモジュール，あるいはストラテジーが存在しないか検索される．システムは状態確認のタスクを持つストラテジー“ AlterCondChecker ”を持っており，このコンポーネントが検索により発見され，構成の変更によるコンポーネント故障への適応が期待される．ストラテジー“ AlterChecker ”は記憶されている現在の状態を直接取得するのではなく，機体に搭載された高度センサーと加速度センサーの情報から機体が飛行中であるかを推定し，現在の状態を判断するストラテジーである．現在の状態を直接取得しないため，コンポーネントの故障に関わらず動作するが，処理工程が増えるため状態確認タスクに必要な時間が増加してしまう．しかし，システム設計段階においてストラテジー“ AlterChecker ”による状態確認タスクの代替による処理時間増加に影響される離陸や着陸，充電機能には，このような処理時間増加をある程度許容できるような時間制約の設定がなされているため，タスクの代替によって故障への適応がなされるはずである．

シナリオ 2．モーターの劣化：機体には 4 つのプロペラが搭載されており，それぞれ独立したモーターによって回転駆動されている．モーターの劣化により，モーターの起動時間が増加することによる時間増加によって，表 2 にて許容遅延が赤文字で記された時間制約の厳しい機能において制約が満たされなくなる．このような，実行系列を完遂したが時間制約が満たされなかった場合を想定する．

処理時間の更新などにより時間制約が満たされなくなると，構成の変更により制約を満たすことができるような構成を作ることができるかどうかの検索が実行される．検索時には，時間制約が満たされていない機能のループを深さ優先で辿り，タスクを他のストラテジーやモジュールによって代替することにより時間制約を充足できるかどうかが試行される．水平移動や回転，姿勢制御などプロペラの回転数を変化させることによって機体を移動させる場合，それぞれの移動に合わせて各モーターの回転数を算出し，実際にモーターの回転数を変化させるという 2 段階の処理が行なわれる．2 つの処理はそれぞれ初期構成においてモジュール“ SpeedCalculater ”とモジュール“ Motor ”が担当している．このシステムでは，初期構成のモーターに代わることのできる駆動部品は搭載されておらず，プロペラ回転のタスクを代替することができない．しかし，このシステムは“ SpeedCalculater ”と同じタスクをもつモジュール“ RoughCalculater ”を持っている．このモジュールは異なる算出アルゴリズムを用いることにより，算出数値の精密さは劣るが算出にかかる時間を大きく低減させたコンポーネントに対応しているものである．よって，モーター回転数算出のタスクをモジュール“ RoughCalculater ”が代替することにより，モーター劣化による処理時間増加への適応が期待される．

4-4 実験結果

図 6 左 はシナリオ 1 実行時のログである．実行ログの 7 行目までは，直前のコマンド実行処理終了後の時間制約充足判定が行なわれている．8 行目にてシナリオ 1 を実行し，意図的に故障を注入している，続い

て 10 行目にて故障検出時の動作を確認するため，状態確認タスクが含まれる着陸機能が実行されるよう着陸コマンドを送信している．そして 17 行目にてコンポーネントの故障によって実行系列が中断されたことが検出されている．それに対して，18～21 行目にて適応動作が起き，問題が解決されたことが読み取れる．結果，22 行目以降にて実行できなかった処理が再度実行され，送信した着陸コマンドによって期待される動作が得られている，また，適応動作後のシステム構成を確認したところ，状態確認タスクを担っていたモジュール“ CondCheck ”がストラテジー“AlterCondCheck”に差し代わっていることが確認できた．これらの結果から，シナリオ 1 のシミュレーションについては期待する適応動作が実現されていると言える．

図 6 右 はシナリオ 2 実行時のログの一部を示したものである．いくつかのコマンドを送信，実行を確認した後に 9 行目にてシナリオ 2 を実行し，意図的にモーター回転タスクの処理時間を増加させた．その後，9 行目にて上昇コマンドを送信したことにより，28 行目まで上昇のための処理が実行されていることが分かる．そして 29 行目のコマンド実行処理終了後の時間制約充足判定にて，満足されていない制約が存在していることが 32 行目から分かる．そのため，33 行目からシステムの再構成の検索が実行されている．その結果 37 行目までの適応動作で時間制約を満足するシステムの構成が発見され，問題が解決されている，また，適応動作後のシステム構成を確認したところ，回転数算出タスクを担っていたモジュール“ SpeedCalculater ”が“ RoughCalculater ”に差し代わっていることが確認できた．これらの結果から，シナリオ 2 についても期待する適応動作が実現されていると言える．

シナリオ 1 は制約違反の原因となったコンポーネントを使用しないシステム構成をとることにより時間制約を満足した例である．また，シナリオ 2 は制約違反の原因となったコンポーネントがシステム構成に欠かせない場合であっても，関連コンポーネントの再構成によって時間制約を満足した例である．2 つのシナリオを用いたシミュレーション実験を通して，提案する動的検証メカニズムを用いて実装されたシステムが時間制約の動的検証により時間制約違反を検出し，期待された適応動作をみせることが確認された．このことから，本調査研究で設計した動的検証メカニズムは時間制約の考慮が必要な自己適応システムの実現に有効であると言えよう．

```
1   Battery 85%..
2   ModeController:
3       normalmode
4 //////// xml file generating ... ///////
5 //////// generation completed ///////
6 verify completed
7 queries are sutisfied
8 sim1
9 execute simulation 1: CondiCheck module trouble
10 landing
11 execute processCommandDecode
12 Reciever:
13  command receive
14 Decoder:
15 execute processLanding
16 CondChecker:
17  CondCkeck Module is brokendown
18 //////// xml file generating ... ///////
19 //////// generation completed ///////
20 verify completed
21 trouble is solved
22 retry process Landing
23 execute processLanding
24 AltitudeSensor:
25  Altitude 50
26  condition: flying
27 GyroSensor:
28  collecting data..
29 Moter:
30  rotate motors
31 Stop:
32  stop motors
33 ConditionController:
34 change condition: staying
35 execute processCheckBattery
36 //////// xml file generating ... ///////
37 //////// generation completed ///////
38 verify completed
39 queries are sutisfied
40 exit
```

```
1  Battry:
2      Battery 70%..
3  ModeController:
4      normalmode
5  //////// xml file generating ... ///////
6  //////// generation completed ///////
7  verify completed
8  queries are sutisfied
9  sim2
10 execute simulation 2: Motor deterioration
11 processing time of Motor is updated
12 moveup
13 execute processCommandDecode
14 Reciever:
15      command recieve
16 Decoder:
17 execute processMoveVertical
18 Direction:
19  Direction up
20  AltitudeSensor:
21      Altitude 50

        --- 中略 ---

22      rotate motors
23      hovering..
24 execute processCheckBattery
25 Battry:
26      Battery 55%..
27 ModeController:
28      normalmode
29 //////// xml file generating ... ///////
30 //////// generation completed ///////
31 verify completed
32 queries are not sutisfied
33 try reconfiguration
34 //////// xml file generating ... ///////
35 //////// generation completed ///////
36 verify completed
37 reconfigure completed
38 exit
```

図 6　各シナリオの実行ログ（左：シナリオ 1，右：シナリオ 2）

5 まとめ

本調査研究では，実世界を対象とする組み込み機器上で動作可能な自己適応ソフトウェアのプログラミングフレームワークのプロトタイプを構築し，また，時間制約を考慮可能な動的検証メカニズムを設計，実装し，その有効性を評価した．プログラミングフレームワークについては，研究代表者が先行研究で提案しているフレームワークを軽量化することで，組み込みシステムなどのリソースに制約のある実行環境での利用可能性を追求した．また，時間制約を考慮可能な動的検証メカニズムについては，時間拡張された状態遷移モデルを用いた設計とモデル検査ツール UPPAAL の検証器を利用することによりシステム実行時の動的な時間制約検証を実現した．また，小型無人航空機制御システムを題材としたシミュレーション実験により，時間制約の考慮によりシステムの振る舞いが動的に変更されることを確認し，同メカニズムの有効性を評価した．今後は，軽量フレームワークと時間制約を考慮可能な動的検証メカニズムとを統合することで，実世界上での自己適応システムを構築するための統合的プログラミングフレームワークを構築し，同フレームワークを用いた実証実験を実施する予定である．

【参考文献】

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," ACM Transaction on Autonomous and Adaptive Systems, vol.4, no.2, pp.14:1-14:42, may 2009. http://doi.acm.org/10.1145/1516533.1516538

[2] M.C. Huebscher and J.A. McCann, "A survey of autonomic computing - degrees, models, and applications," ACM Computing Surveys (CSUR), vol.40, no.3, pp.7:1-7:28, aug 2008. http://doi.acm.org/10.1145/1380584.1380585

[3] 中川博之，大須賀昭彦，本位田真一，“ビヘイビア記述に基づく自己適応システム実装フレームワークの提案”，人工知能学会論文誌，Vol.26，No.1，pp.1-12（2011）．

[4] H. Nakagawa, A. Ohsuga, and S. Honiden, "Towords dynamic evolution of self-adaptive systems based on dynamic updating of control loops," 2012 IEEE Sixth International Conference on Self-Adaptive and self-Organaze System (SASO'12), pp.59-68, 2012.

[5] Telecom Italia, “JADE: Java Agent Development Framework”, http://jade.tilab.com/

[6] Lego, "mindstorms ev3", https://www.lego.com/ja-jp/mindstorms/about-ev3.

[7] M.U. Iftikhar and D. Weyns, "Activforms: Active formal models for self-adaptation," Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'14), pp.125-134, ACM, New York, NY, USA, 2014. http://doi.acm.org/10.1145/2593929.2593944

[8] A. Filieri, G. Tamburrelli, and C. Ghezzi, "Supporting self-adaptation via quantitative verification and sensitivity analysis at run time," IEEE Transactions on Software Engineering, vol.42, pp.75-99, 2015.

[9] N. Esfahani, A. Elkhodary, S. Malek, "A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. 39, No. 11, pp. 1467-1493, 2013.

[10] V.E. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos, "Requirements-driven software evolution," Computer Science, vol.28, no.4, pp.311-329, Nov. 2013. http://dx.doi.org/10.1007/s00450-012-0232-2

[11] H. Tsuda, H. Nakagawa, and T. Tsuchiya, "Towards self-adaptation on real-world hardware: A preliminary lightweight programming framework," 2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems (SASO'15), pp.176-177, IEEE, 2015.

[12] K.G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," International Journal on Software Tools for Technology Transfer, vol.1, no.1-2, pp.134-152, 1997.

〈発　表　資　料〉

題　　名	掲載誌・学会名等	発表年月
時間制約を考慮可能な動的検証機能を備えた自己適応システム実装フレームワーク	第 15 回情報科学技術フォーラム（FIT2016），第 1 分冊 pp. 41-48	2016 年 9 月
A Dynamic Verification Mechanism for Real-time Self-adaptive Systems	The 10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2016), (Poster Session), pp. 265-266	2016 年 9 月
Caching Strategies for Run-time Probabilistic Model Checking	The 11th International Workshop on Models@run.time (MRT 2016), pp. 1-8	2016 年 10 月
システム環境の変化傾向に基づいた効率的な構成切り替え手法	電子情報通信学会 知能ソフトウェア工学研究会(SIG-KBSE)，KBSE2016-39, pp. 1-6	2017 年 3 月
キャッシュの効率利用による自己適応システムの動的モデル検査法改善	電子情報通信学会 知能ソフトウェア工学研究会(SIG-KBSE)，KBSE2016-40, pp. 7-12	2017 年 3 月