ソフトウェア工学 134－15
（２００１．１０．１９）

リアクティブシステムの安全性に関わるソフトウェアパターンの研究

伊藤清人，小林洋（東海大学）
kobayasi@keyaki.cc.u-tokai.ac.jp

あらまし

オブジェクト指向ソフトウェア開発の分野では，ソフトウェアパターンの研究が盛んに行われているが，システムの信頼性や安全性を目的としたパターンの研究はあまり見当たらない．本研究では，リアクティブシステムの安全性のためのソフトウェアパターンについての検討を行う．はじめに，リアクティブシステムがイベントの刺激と反応として表せることに着目し，イベントの順序と時間制約条件をチェックするためのイベントチェッカと名づけた安全機構を既存のパターンを活用しながら構築する．次に，この安全機構の有効性を，鉄道関係の二つのモデルを用いて検討する．

キーワード

ソフトウェアパターン，ソフトウェアの安全性，イベントチェッカ，リアクティブシステム

A Software Pattern for Safety in a Reactive System.

Kiyohito Itoh, Hiromi Kobayashi(Tokai University)

Abstract

Software patterns are widely studied, however, there are few patterns for safety or dependability. This paper examines a software pattern for safety in a reactive system. We construct a mechanism which we call an event checker including several known software patterns to check order of events and their timing constraints.

Examples of implementation to railroad models are shown to examine the usefulness of this pattern.

Keywords

software pattern, software safety, event checker, reactive system

１．はじめに

システムの信頼性の立場からは，ソフトウェアはハードウェアと異なり，正しく作られたものが劣化により故障（Failure）を起こすことは通常は考えられない．しかしながら，仕様書の誤りに起因するものやプログラムミスによるものを含め，プログラムのバグを全て排除することが現実的に不可能であるため潜在的なエラーが顕在化したり，また，ネットワーク等を通じて外部から侵入したウイルスによりプログラムが書き換わってしまうことが有り得るため，これらを起因とするソフトウェアによる故障は起こり得る．システムの安全性（Safety）[1]のためソフトウェアに起因する故障に対処する方法として，ハードウェアの予備切替方式と同様なリカバリブロック（Recovery Block）方式[2]や多数決方式と同様なＮバージョンプログラミング（N-Version Programming）[3]のように，同じ機能を持つソフトウェアコンポーネントを複数用意する方法が提案されている．しかしながら，同じ機能を持つコンポーネントを複数個開発するということは，費用対効果の点から極めて高い信頼性を要求されるシステムを除き用いられてはいない．

別の方法として，リアクティブシステム[4]，つまり工場における生産ラインのシーケンス制御システムのように，イベントによる刺激とその反応として捉えることのできるシステムでは，制御コンピュータに送信されるイベントの順序と時間制約条件をチェックするための機構（以後，イベントチェッカと呼ぶ）によりシステムの故障を検出し対処することが考えられる．この方式は，トレースアサーション手法（Trace Assertion Method）として文献[5]で提案されているイベントの順序列に基づくシステムの挙動のチェックをイベントチェッカとして実現しようとするものである．イベントチェッカはシステムが正常に機能しているのであれば冗長である．しかしながら，制御コンピュータのプログラム内に潜在するかもしれないバグ等に起因するイベントの順序や時間制約条件の異常をチェックするという意味からは有用である．この場合，イベントチェッカが費用対効果の点で作成が容易で誤りが入りにくい表現形式であることと，制御コンピュータのプログラム自体の変更をできるだけ抑え，外部からはイベントチェッカの存在を気にする必要がないようなプログラムの構造であることが重要である．そこで我々は，既存のいくつかのソフトウェアパターン[6]を用いてイベントチェッカを作成した．イベントチェッカ自体一つのソフトウェアパターンと見なすことができる．なお，本研究で対象とするのは時間を扱うシステム（timed system）ではあるが厳密な意味でのリアルタイムシステムではない．以下，２．では安全機構の構造と期待される効果を示し，３．では安全機構を実現するための既存のパターンの適用を示す．４．ではイベントの順序と時間制約条件をチェックする仕組みを説明し，５．では鉄道関係の二つのモデルへの適用を示す．

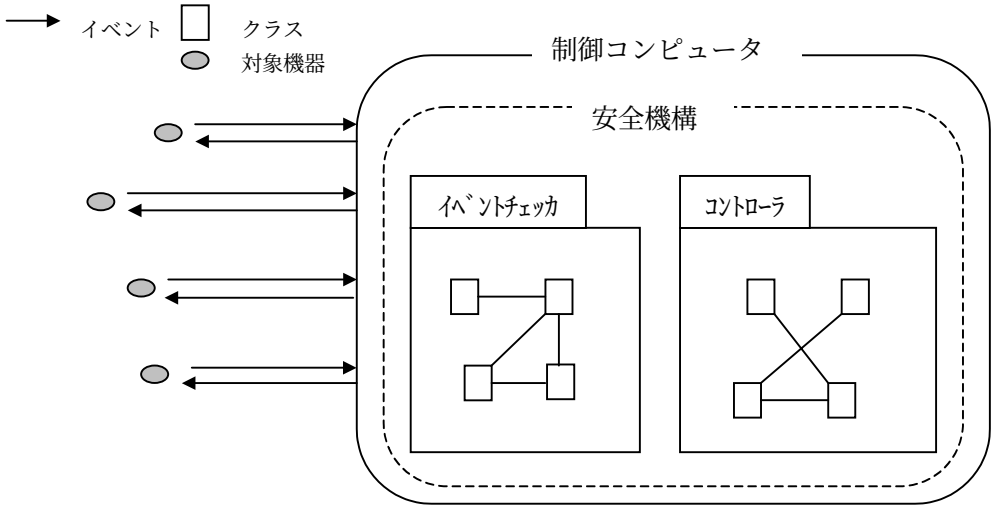


図１　安全機構

2．安全機構

リアクティブシステムがイベントの刺激に対して反応するシステムとしてモデル化できることに着目し，制御コンピュータに送信されるイベントの順序と時間制約条件をチェックする仕組みを制御コンピュータに組み込むことにする．

2.1　構造

この安全機構は，コントローラパッケージとイベントチェッカパッケージにより構成される（図1）．コントローラパッケージは，制御コンピュータの対象機器を操作するための責務が割り当てられたクラス群により構成される．イベントチェッカパッケージは，制御コンピュータに送信されるイベントに関して，作業のシナリオにより規定されるイベントの順序と時間制約条件を満たしているかをチェックするための責務が割り当てられたクラス群により構成される．

2.2　期待される効果

- 安全機構を組み込んだ制御コンピュータは，コントローラパッケージとイベントチェッカパッケージでイベントの順序と時間制約条件を２重にチェックできる．
- イベントチェッカパッケージでは，イベントの順序と時間制約条件のチェックに専念できるため，その表現が単純になり誤りが入りにくくなる．

3．ソフトウェアパターンの導入

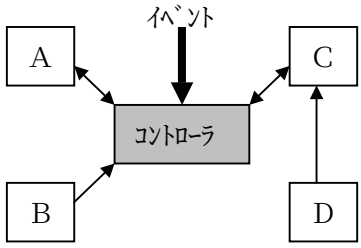
上述した安全機構の実現に当たって生じる問題を解決するために, 以下のような E. Gamma らが示したパターン[6]を適用する．

3.1　Decorator パターンの適用

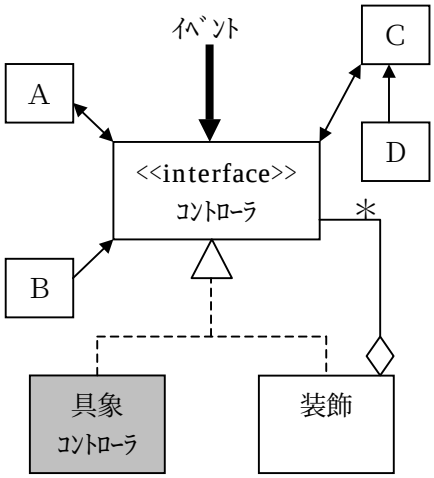
3.1.1　問題

安全機構を実現するには，制御コンピュータに送信されるイベントからチェックの対象とするイベントを抽出する仕組みが必要である．

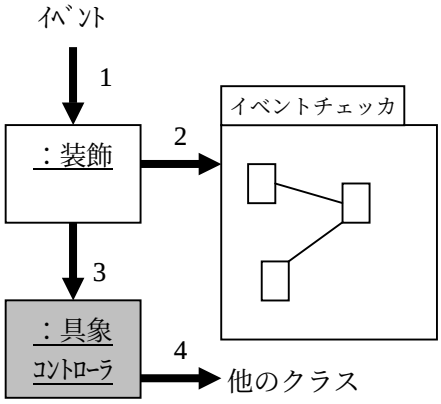
基本的にコントローラパッケージは，制御の対象機器を操作するためにイベントドリブンの機構となっている．したがって，イベントチェッカパッケージは，コントローラパッケージに送信されるイベントからチェックの対象とするイベント（以後，チェック対象イベント）を抽出することになる．しかし，コントローラパッケージのクラス群は，対象機器の制御という本来の責務を果たすよう設計されており，そのクラス構造に対する変更は，新たなバグを生み出す可能性があるため避けたい．したがって，コントローラパッケージのクラス構造に対する影響をできるだけ抑え，チェック対象イベントを抽出するような仕組みが必要となる．



(a)　Decorator パターン適用前



(b)　Decorator パターン適用後



(c)　ｲﾍﾞﾝﾄの流れ

図 2　Decorator パターンの適用

3.1.2　解決

Decorator パターンを適用することにより，イベントチェッカパッケージは，コントローラパッ

ケージに送信されるイベントからチェック対象イベントを抽出することができる．そこで，コントローラパッケージのクラス群において，チェック対象イベントを受け付ける責務が割り当てられた Facade クラスに対して，イベントチェッカパッケージのクラスがチェック対象イベントを抽出できるよう Decorator パターンにより装飾を施す（図 2）．ここで(a)では，コントローラクラスを Facade クラスとしている．そして(b)は，Decorator パターン適用後のクラス構造である．(a)のコントローラクラスを(b)の具象コントローラクラスに置き換え，(a)のコントローラクラスの他のクラスに対する契約を示すコントローラインタフェースを設ける．そのインタフェースを実装する装飾クラスは，コントローラインタフェースへの参照を保持している．結果として(b)のクラス構造は，(c)のようにイベントを送信できるオブジェクト構造を生成することができ，装飾オブジェクトからイベントチェッカパッケージのオブジェクトへイベントが送信されることになる．したがって，(a)のコントローラクラスに依存していたコントローラパッケージの他のクラスは，実装を変更することがなくコントローラインタフェースに依存することになり，コントローラパッケージへの変更の影響を Facade クラスのみに抑えることができる．

3.2　Composite パターンの適用

3.2.1　問題

安全機構を実現するには，制御コンピュータ内に表現されたイベントの順序をチェックする必要がある．作業のシナリオにより規定されるイベントの順序は，対象とするシステムによっては単純な系列とは限らない．したがって，単純な系列ではない場合のイベントの順序を表現する方法と，その表現されたイベントの順序を制御コンピュータ内に取り入れる仕組みが必要になる．

3.2.2　解決

制御コンピュータに取り込む前の段階では，イベントの順序を見やすく表現するため，イベントの順序は，イベント名と分岐／繰り返しを表す制御記号と共にテキストファイル形式のイベントリストとして表現することにする．

制御コンピュータ内では，このイベントリストを解釈することにより生成されるイベントオブジェクト群によりイベントの順序が表現される．すると，イベントリストは，制御記号をノードとする再帰的な階層構造で表現できる．ここで Composite パターンを適用することにより，イベントクラスと制御記号クラスのオブジェクト群からなる階層構造を表現できる（図 3）．図 3 の制御記号クラスは，スーパークラスである要素クラスへの参照（複数）を保持しているため，要素クラスのサブクラスであるイベントクラスと制御記号クラスのオブジェクトを保持することが可能となる．

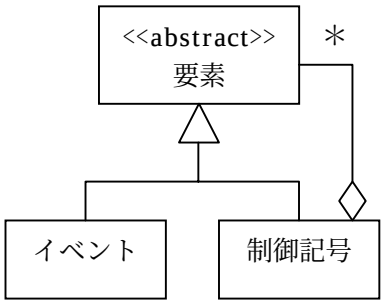


図 3　Composite パターンの適用

3.3　State パターンの適用

3.3.1　問題

安全機構を実現するには，イベントの時間制約条件をチェックする必要がある．イベントの時間制約条件をチェックする際には，あるイベントが生じてから次のイベントが生じるまでの時間が最小遅延時間（以後，最早時間）から最大遅延時間（以後，最遅時間）に収まるかを調べる必要があり，その表現方法が問題となる．

3.3.2　解決

イベントの時間制約条件は，あるイベントからの経過時間により，最早時間に満たない場合，許容された時間の範囲内にある場合，最遅時間を超えている場合に分類できる．制御コンピュータ内では，これらの時間制約に関する場合分けを状態（以後, 時間制約状態）により表現する．しかし，イベントクラスは，自身のみで各時間制約状態を表現するとクラスの凝集性が低く内部構造が複雑なクラスになる．そこで，State パターンを適用することにより，イベントクラスからイベントの時間制約条件チェックの処理を分離できる（図 4）．イベントクラスは，状態インタフェースを実装したクラスに対して，時間制約状態に依存する処理を委譲する．委譲先の時間制約状態クラスは，自身の状態として表現する時間制約条件をチェックし, イベントクラスの時間制約状態を遷移させる．

これにより，イベントの時間制約条件を表現することができる．

４．イベントチェッカパッケージ

前節で示した解決方法をふまえてイベントチェッカパッケージにおけるイベントの順序と時間制約条件をチェックする仕組みを説明する．

4.1　構造

イベントチェッカパッケージは，解釈パッケージと要素パッケージ，イベントチェッカクラスにより構成される（図 5）．図 5 では，イベントチェッカパッケージ全体を理解できるよう各パッケージ内のクラスの一部のみ記述してある．解釈パッケージは，作業のシナリオにより規定されるテキストファイル形式のイベントリストを解釈しイベントオブジェクト群を作成するための，イベントリスト解釈クラスを中心としたクラス群により構成される．要素パッケージは，イベントチェッカパッケージに必要な要素であるイベントと制御記号を表現するクラスにより構成される．そして，イベントチェッカクラスは，イベントチェッカパッケージ全体を制御する責務を持つ．

4.2　イベントの順序チェック

イベントの順序チェックは，コントローラパッケージの装飾クラスがイベントチェッカパッケージのイベントチェッカクラスの check()メソッドを利用して行う．このメソッドでは，引数として指定されたイベント名が作業のシナリオにより規定されるイベントの順序として正しいかを判定する．判定には，イベントチェッカオブジェクトに現時点で設定されているイベントオブジェクトの getNextEvent()メソッドを利用して，イベント名に該当する次のイベントオブジェクトの有無により行う．

4.3　イベントの時間制約条件チェック

イベントクラスでは，時間制約条件をチェックするために，start()メソッドにより開始時点，getNextEvent()メソッドにより終了時点が示される．両メソッドは，呼ばれた時点で自身に設定されている時間制約状態オブジェクトにイベントの時間制約条件チェックの処理を委譲する．イベントオブジェクトでは，初期状態として最早時間に満たない状態オブジェクトが設定される．イベントオブジェクトの start()メソッドが呼ばれる

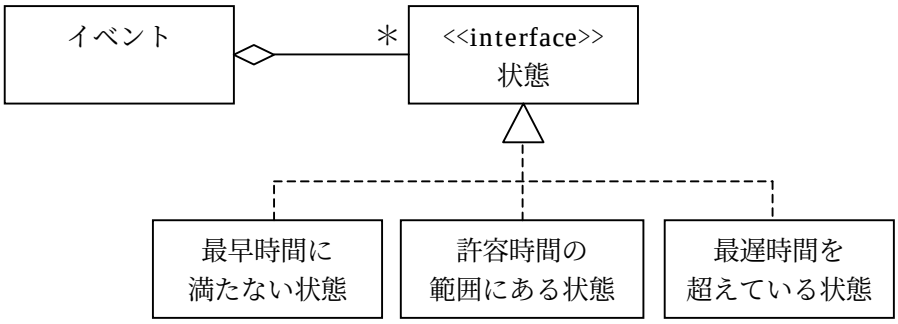


図 4　State パターンの適用

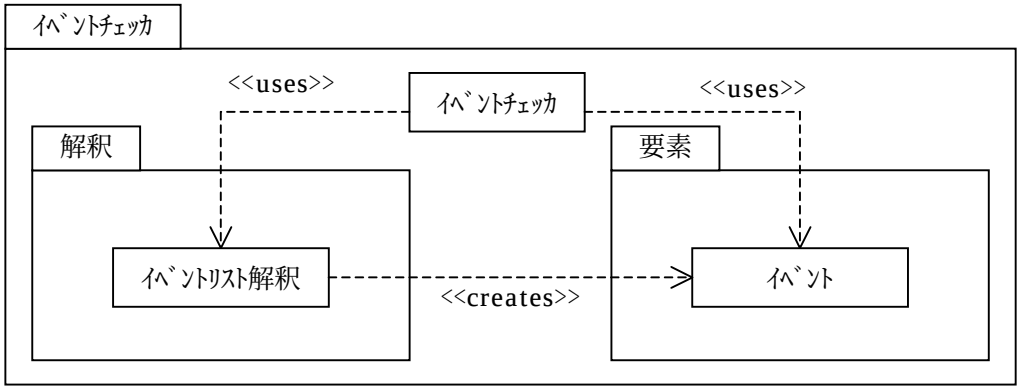


図 5　イベントチェッカパッケージ

と，最早時間に満たない状態オブジェクトは，イベントの経過時間が自身に設定された最早時間に満たないかをチェックする．最早時間が経過すると，最早時間に満たない状態オブジェクトは，許容時間の範囲にある状態に遷移する．許容時間の範囲にある状態オブジェクトも同様に，自身に設定された許容時間（最遅時間－最早時間）にイベントの経過時間が収まるかをチェックする．許容時間が過ぎると，許容時間の範囲にある状態オブジェクトは，最遅時間を超えている状態に遷移する．このように，イベントの時間制約条件チェックは，イベントの時間制約状態を表現するオブジェクトの状態遷移により行われる．

５．適用例

イベントチェッカを鉄道関係の２つのモデルに適用して，その有用性を示す．

5.1　踏切モデル

リアクティブシステムの例題として，しばしば用いられる踏切モデル[7,8]をまず取り上げる．踏切制御システムは，「列車」「踏切」「信号機」「センサ」「コントローラ」から構成されているとする．鉄道の踏切において，列車が踏切から一定距離に接近するとセンサはそれを検知し，コントローラに伝える．コントローラは踏切にゲートを下げるよう指示を送り，ゲートが閉じる．列車が踏切を通過すると別のセンサがそれを検知し，コントローラに伝える．コントローラは踏切にゲートを上げるよう指示を送り，ゲートが開く．

このモデルのコントローラに安全機構を適用する．イベントの順序と時間制約条件を規定するためには，１回のゲート開閉動作における最大通過列車数（反対方向の列車数も含める）を決める必要がある．これは，最大通過列車数に応じてゲートを開くタイミングが異なるためである．今回は，最大通過列車数を２と仮定した．図 6 では，イベントを記号とその識別番号により，イベントの順序関係を矢印により示している．破線の矢印は，さらにこの先の番号のイベントに順序が続くことを示している．番号１→２→３→４は，通過列車数が１本の場合に生じるはずのイベントの順序を示している．通過列車数が２本の場合には，２本目の通過列車が検知されるタイミングにより，先の番号１→２→３→４又は番号５，７，10 の何れかをたどるイベントの順序になる．したがって最大通過列車数を２と仮定した場合，イベントの順序は４つのシナリオにより表現される．これによりイベントの順序とイベント時間制約条件（図 6 では省略）を満たさないイベントのチェックを可能にした．また，コントローラパッケージ内のクラス群に対する影響は，3.1 節で述べたように，Facade クラスのみに抑えることができた．

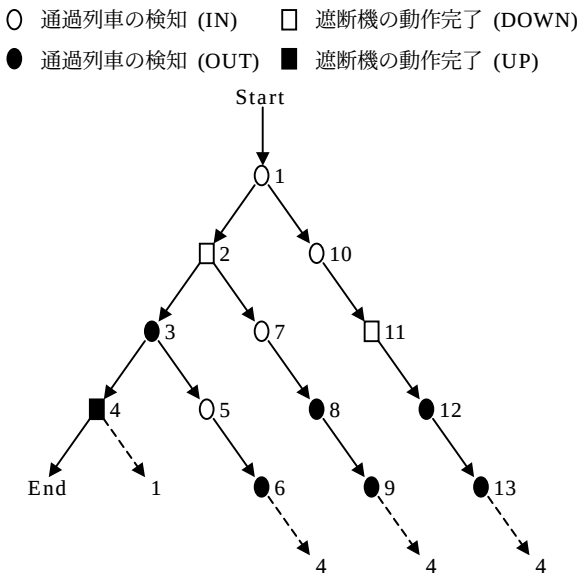


図 6　イベントの順序

5.2　連動装置

駅構内において列車を安全に運行させるために，ポイントと信号機を制御する連動装置[9]について考えてみる．信号機の進行制御は，信号機が保証する進路上に列車がいないこと，ポイントが正しい方向を向いていること，及び競合する他の進路の信号機が進行を示していないことを確認して行われる．ポイントの制御は，信号機が進行現示の間及び進路上に列車がいる間は，ポイントが転換しないよう鎖錠される．

このモデルでは，連動装置に安全機構を適用する．連動装置では，ポイントと信号機を制御するために，ある区間のレールを表す軌道区分から列車の位置情報をイベントとして取得する．そのイベントは，通過列車を検知した場合にも連動装置に送信される．今回は，この通過列車の検知を表すイベントをチェックする．イベントの順序と時間制約条件は，連動装置への進路指示とその指示された進路を構成する軌道区分からの通過列車の検知により規定される（図 7）．例えば，A1-A3 は進路イベントで，A1 から A3 に列車を運行する指示が生じることを意味する．また，A1 は通過

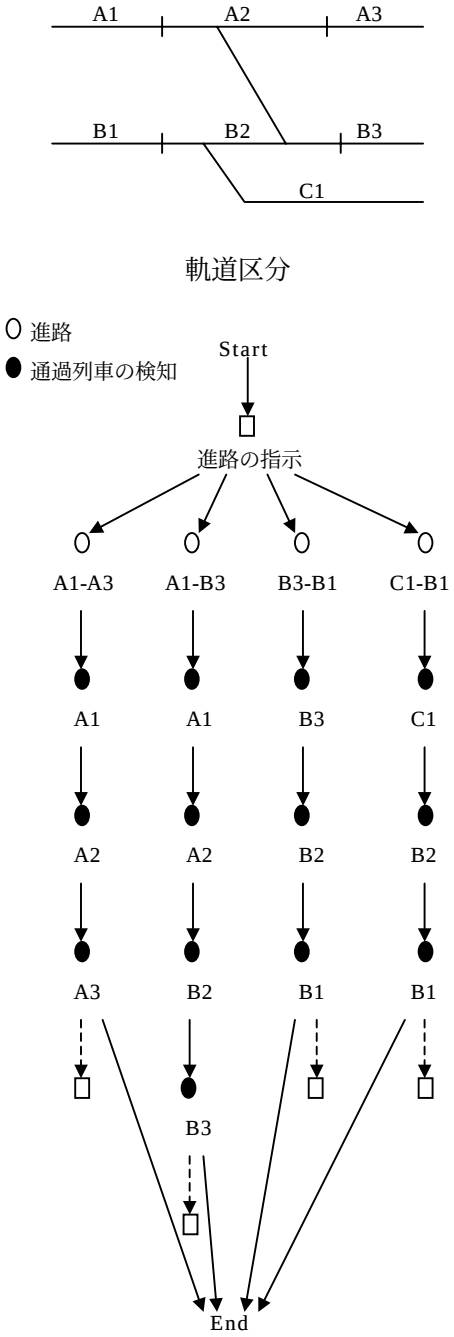


図 7　軌道区分とイベントの順序

列車の検知イベントで，軌道区分 A1 からのイベントであることを意味している．したがってイベントの順序は，進路に応じたシナリオにより表現される．進路の指示イベントを設けているのは，シナリオを構成する最後のイベント（通過列車の検知）に続くイベントが複数に分かれるのを防ぎ，イベントの順序を理解しやすくするためである．これによりイベントの順序と時間制約条件（図 7 では省略）を満たさないイベントのチェックを可能にした．また，コントローラパッケージに対する影響は，Facade クラスのみに抑えることができた．

6．おわりに

本研究では，リアクティブシステムにおいて，イベントチェッカを組み込み安全機構を実現するためのソフトウェアパターンの検討を行い，鉄道関係の二つのモデルに実装を行った．

本研究では時間を扱っているが，厳密な意味でのリアルタイム性は扱っていない．プロセッサの実行時間やプロセスの実行スケジューリング，イベントの並行性（Concurrency）までは扱っていないためである．システムによっては，イベントの順序をチェックする際には並行性，つまり，ある時間区間内で生起する複数のイベントの順序が規定できない場合について考慮しなければならないことがある．しかしながら，全てのイベントをチェックするのではなく，安全上重要と考えられるイベントのみを抽出してチェックすることにより，この問題は解消できると考えられる．全てのイベントのチェックでなくても，イベントチェッカは有用と考えられる．

イベントチェッカにおいてイベントを解釈する部分については，コンポーネント化すれば他のシステムにおいても使うことができる．イベントと時間制約については，テキストファイル形式のイベントリストに分岐／繰り返しを表す制御記号と共に記述するだけであるので，作成や変更が容易である．また，処理の詳細まで記述してある制御コンピュータ側のプログラムと異なりイベントと時間制約が見やすく，誤りが入りにくいと考えられる．

イベントチェッカのアイディア自体は，冗長な構成法としてではないが，トレースアサーション手法で提案されており，ハードウェアの動作チェックに用いられている例があるようだが，ソフトウェアパターンとしては我々の知る限り提案されていない．

本来パターンは，数多くの実際の開発経験から発見するものと言われているが，この研究では，既に提案されているアイディアを基に安全機構の構築を試みるというアプローチを採った．我々がイベントチェッカとして示したクラスの集合とその関連は，いくつかのリアクティブシステムにおけ

るある種の安全性を実現するための１つのパターンと呼べるかもしれない．しかしながら，パターンと呼べるだけの一般性が不足しているようにも思える一方，リアクティブシステムの安全性については，要求されるリアルタイム性や安全性により異なるパターンがあると考えられる．この分野についての更なる研究が必要と考えられる．

謝辞　鉄道の連動装置についての疑問に回答して下さいました鉄道総研の松本真吾殿に深く感謝致します．

参考文献

[1] N.G.Leveson : ”Safeware : System Safety and Computer,” Addison-Wesley, 1995.
[2] B.Randell : “System structure for software fault tolerance,” IEEE Trans. Software Engineering, vol.SE-1, No.1, pp.220-232, 1975.
[3] A.Avizienis and L.Chen : “On the implementation of N-version programming for software fault tolerance during execution,” Proc.COMPSAC, pp.149-155, 1977.
[4] D.Harel : “Statecharts : A visual formalism for complex systems,” Science of Computer Programming, Vol.8, pp.231-274, 1987.
[5] W.Bartussek and D.L.Parnas : “Using assertions about traces to write abstract specifications for software modules,” Proc.2nd Conference European Cooperation in Informatics on Information Systems Methodology, pp.211-236,1978.
[6]Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, 本位田 真一, 吉田 和樹 監訳 : “オブジェクト指向における再利用のためのデザインパターン,” ソフトバンク, 1995.
[7] N.G.Leveson and J.L.Stolzy : “Safety analysis using Petri nets,” IEEE Trans. Software Engineering, Vol.SE-13, No.3, pp386-397, 1987.
[8] R.Alur and D.Dill : “The theory of timed automata,” Real-time theory in practice, LNCS Vol.600, p.45-73, Springer-Verlag, 1992.
[9] 松本 真吾 : “連動装置の安全性仕様と自動検証,” 電子情報通信学会 信学技報, p.29-36, 1999.