Vol.2009-EMB-14 No.1
2009/7/24

HAZOP 分析による ソフトウェア異常動作検出条件の導出手法の提案と実装

日高隆博[†1]　山崎二三雄[†1]　中本幸一[†3]
本田晋也[†2]　高田広章[†2]

車載ソフトウェアの大規模化に伴って，従来型開発手法による安全性検証が困難となってきている．本研究では，安全性分析手法である HAZOP を用いてソフトウェア異常検出条件を導出する手法について提案する．また，この異常検出条件を用いたソフトウェア異常監視機構について実装を行い，本手法の有効性について評価を行う．

本手法は特にコンポーネント機構を用いたソフトウェアに対して有効であり，また，組込みソフトウェアの制約に合わせた監視条件の設定が容易であることを示す．

A Method of Deriving Anomaly Detection Rule by HAZOP Analysis

Takahiro Hidaka,[†1] Fumio Yamazaki,[†1]
Yukikazu Nakamoto,[†3] Shinya Honda[†2]
and Hiroaki Takada[†2]

With enlargement the scale of automotive systems, it becomes harder to verify by traditional method. In this study, we propose to derive anomaly detection rule from result of HAZOP. And we evaluate this method by example and implementation.

We show this method is valid for component oriented software, and easy to adopt rule for constraint of embedded system.

1. はじめに

従来型の車載システムにおいては徹底したコードレビューによって安全性を確保してきていたが，車載システムの大規模化に伴って安全性の確保が困難となってきている．この問題に対処するためのひとつの方法として，AUTOSAR に代表されるコンポーネント機構の導入がある．

コンポーネント機構を導入することによって，コンポーネント単位での仕様の策定が行われ，それに従って仕様検証・開発・テストが行える．コンポーネント単位で各作業が分業できるために開発効率が向上することが期待される反面，安全性検証については，従来どおりに全てのソースコードをレビューすることによって行うのみではコンポーネント化のメリットを生かすことができない．この問題を解決するために，FMEA，FTA，HAZOP など[1)]のさまざまな安全性評価手法が取り入れられてきている．

また，現在の車載システムなどの組込みシステムにおいては，機能安全の考え方が導入されはじめている．機能安全の考え方では，危険側故障の発生頻度の許容上限を定め，その発生頻度を下げるためにハードウェアやソフトウェアによる実行環境における自己診断・自己監視機能を導入することが行われる．しかし，現在のところ汎用的な自己監視機能としてはウォッチドッグなどの単純な機構しか存在しないため，より複雑な自己監視機能については個々の車載システムごとにアドホックな実装に頼らざるを得ず，開発やテストにおける工数負担となっている．

これらのことを背景として，われわれは組込みシステムのなかでも特に制御系システムおよび制御系システムと連携して動作するような情報系システムを対象としたソフトウェアによる自己監視機構について研究を進めている．自己監視機構を導入することによって，多数のコンポーネントからなるシステムの安全性確保を容易にし，この自己監視機構を実行環境上のミドルウェアとして組み込むことで，機能安全対応コンポーネント機構を実現することを目標としている．

本研究では，安全性分析手法のひとつである HAZOP を用いて監視条件を導出するため

†1 名古屋大学 大学院情報科学研究科 附属組込みシステム研究センター Center for Embedded Computing Systems, Nagoya University
†2 名古屋大学 大学院情報科学研究科 Graduate School of Information Science Nagoya University
†3 名古屋大学 大学院情報科学研究科 附属組込みシステム研究センター Center for Embedded Computing Systems, Nagoya University / 兵庫県立大学 応用情報科学研究科 Graduate School of Applied Informatics University of Hyogo

Vol.2009-EMB-14 No.1
2009/7/24

表 1 ソフトウェア異常の原因

異常原因		HW 故障	実装誤り	攻撃	波及元
データ	入力	○	-	○	データ入力元動作
	内部	○	○	○	入力データ 動作異常
	出力	○	○	-	内部データ
動作	分岐異常	○	○	-	分岐条件値 (データ)
	スタック破壊	○	○	-	ポインタ値 (データ)

の手法について提案を行い，現在検討している監視機構の一部として実装を行った．

第 2 章ではわれわれが検討しているソフトウェア異常監視機構の概要について説明し，第 3 章で HAZOP を用いた監視条件導出手法についての提案を行う．第 4 章では実装の概要について説明し，第 5 章で評価を行う．第 6 章では本研究と関連した研究について紹介する．

2. ソフトウェア異常監視機構

本章では，本研究の前提となる，ソフトウェア異常監視について整理する．まずわれわれが前提としている用語について定義し，われわれの考えているソフトウェア異常監視機構のモデルについて説明する．

2.1 用 語 定 義

ソフトウェア異常とは，対象ソフトウェアが正常状態として仕様定義された内容とは異なる動作をすることを指す．ソフトウェア異常には外部から観測できる場合と観測できない場合の両方の可能性が考えられる．

ソフトウェアの異常監視とは，ソフトウェア異常を検出し，その異常状態を取り除いて回復を行うことを目的とする．一般には異常監視の機構はソフトウェア，ハードウェア，またはそれらの組み合わせのいずれの手段でも実装することが可能である．

ソフトウェア異常の原因には，ハードウェア故障・内部実装の誤り・外部からの攻撃の 3 種類があると考えられる．これらの原因によって発生する異常は，データの異常または実行系列の異常として観測可能となる．これらの関係について**表 1** のように分析を行った．

データ異常と動作異常は互いに波及しあう関係にあり，ひとつの異常が原因となって別の異常が発生する．また，前述のように全ての異常が観測可能であるとは限らないことから，ソフトウェアの異常からの回復を行うためには，観測した異常から本来の原因を推定し，回復を行う機構が必要となる．

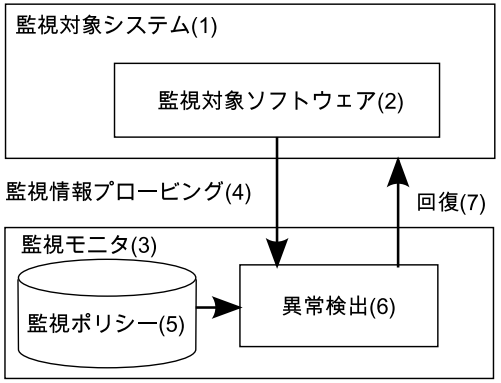

図 1 ソフトウェア異常監視機構

2.2 ソフトウェア異常監視機構

われわれの考えるソフトウェア異常監視機構のモデルについて**図 1** に示す．監視対象システム (1) の構成要素として監視対象ソフトウェア (2) が存在するとき，それらを監視する機構を監視モニタ (3) [⋆1] として実装する．監視モニタでは監視対象ソフトウェア実行環境において監視情報をプロービングする (4)．プロービングによって得られる情報を監視ポリシー (5) に従って解析することによって異常を検出 (6) し，監視対象システムに対して適切な回復を行う (7)．

一般的に，これらの要素 (1)-(7) について定義することによってソフトウェア異常監視機構を定義することができると考えている．

3. HAZOP 分析による異常監視条件の導出

本章では，本研究の主題となる，HAZOP 分析による異常監視条件導出手法についてバックガイドモニタシステムを例題として説明する．

3.1 本研究の特徴

IEC 61882[2] で定義されている HAZOP は，仕様書に記載されたさまざまな定義値に対してガイドワードを適用し，その結果としてもたらされうるハザードと，そのハザードをも

⋆1 このモデルでは監視対象システムと監視モニタは独立した要素としてモデル化しているが，実装上は監視対象システム内部の監視機構として組み込まれるようなことも考えられる．

Vol.2009-EMB-14 No.1
2009/7/24

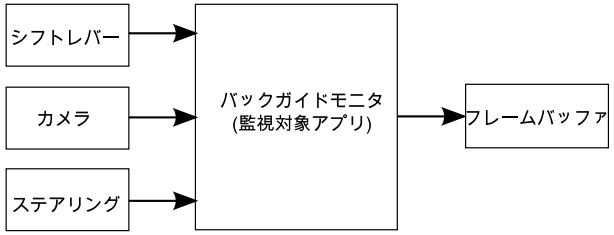

図 2 バックガイドモニタシステム構成

たらす原因の両方について分析を行うものである．主に化学プラント向けに用いられていた手法であるが，ソフトウェアにも同様に適用できる[3] ことを示した研究などが多数存在しており，機能安全規格 IEC 61508[4] への対応のためにも利用され始めている．

通常のソフトウェア HAZOP は設計時の安全性分析手法のひとつであって，分析結果をもとにして設計変更や仕様追加を行うことでシステムの安全性を高めるために用いられている．しかしわれわれは，このソフトウェア HAZOP による分析結果をシステム実行環境における監視条件を決定するために利用できるのではないかと考えた．

監視条件として HAZOP を用いることの利点には，

(1) ハザード分析によって各異常状態ごとの危険度・重大度が分析されており，それがそのまま監視条件としての優先度として利用可能である
(2) ハザードの原因分析結果を利用して異常検出時の回復手順を設計可能である
(3) コンポーネント仕様においては入出力の仕様を定めることが必須であり，その仕様記述を有効に利用できる

という 3 点があると考えている．

3.2 HAZOP 分析例

本研究での具体例として，車載システムのひとつであるバックガイドモニタシステムについて検討する．まず，システムの構成図を**図 2** に示す．

バックガイドモニタアプリは，シフトレバーが R(後退) になっているときにカメラからの入力画像に対してステアリング角度にあわせたガイド枠を描画し，フレームバッファへ出力を行うソフトウェアである．

このシステムについて，ガイドワードとして**表 2** を用いて HAZOP 分析を行う．バックガイドモニタの入出力についてそれぞれ仕様を仮定し，危険度を分析した結果を**表 3** に示す．

たとえばカメラの画像について，入力周期が 1 秒間に 30 フレーム (30fps) であることと

表 2 HAZOP ガイドワード

	適用対象				
ガイドワード	周期	遅延	範囲値	列挙値	構造値
no	データなし	-	データなし	データなし	データなし
less	少ない	-	小さい	-	-
more	多い	-	大きい	-	-
other than	-	-	-	異なるデータ	-
part of	-	-	-	-	欠落
reverse	-	-	-	-	データ化け
early	-	小さい	-	-	-
late	-	大きい	-	-	-

画像データであることの 2 つの仕様を仮定する．このとき，周期に対しては no, less, more のガイドワードを適用することができ，それぞれ，入力がない場合，秒あたりのフレーム数が仕様より少ない場合，多い場合に対応する．このような仕様の逸脱についてシステムの分析を行うことで，原因と結果をそれぞれ推定する．入力異常の原因は通常はそのデータを出力している部品 (コンポーネント) にあり，結果は出力として現れる．ここでは，原因の可能性はカメラおよびカメラと ECU をつなぐ結線の故障にあり，バックガイドモニタアプリではそれらの異常への対応処理を行っていないために結果としては画像の停止，コマ落ち，処理落ちがそれぞれ発生するものだと仮定する．

この例のように，HAZOP では全ての仕様についてガイドワードを順に適用していくことで網羅的に安全性分析を行えることが大きな特徴である．

危険性については，バックガイドモニタでは画像が停止したりガイド表示に異常が発生するような事象は事故の危険性があるために危険な事象となる．一方，一時的なコマ落ちや画像の異常については画像停止と比較すれば危険性の低い事象だと考えることができる．

本研究では，カメラ画像が停止して更新されなくなるような事象とカメラで撮影してから表示されるまでの間に遅延が発生しているような事象について危険度 4 を割り当てた．そのような事象が起きる場合としてはカメラ入力周期とフレームバッファ出力周期へガイドワード no を適用した場合，および，カメラ入力からフレームバッファ出力までの遅延時間に対してガイドワード late を適用した場合の 3 種類の条件がある．

同様に一時的なコマ落ちなどの画像の乱れに危険度 3，シフトレバーの故障などで後方カメラ画像への切替自体に失敗するような場合を危険度 2 とすることで表 3 のような分析結果を得た．

表 3 HAZOP 分析結果

仕様項目	仕様値	ガイドワード	原因	結果	危険度
カメラ	入力データ周期 = 30(fps)	no	カメラ故障 カメラ入力結線故障	画像停止	4
		less	カメラ一時故障	コマ落ち	3
		more	カメラ異常	処理落ち, 遅延	3
	入力値	part of	データ落ち	部分的表示	3
		reverse	データ化け	画像ノイズ	3
		other than	カメラ入力結線故障	他カメラ画像表示	2
ステアリング	-100 < 入力値 < 100	no	センサ故障 センサ結線故障	ガイド表示停止	2
		less	センサ故障	ガイド表示異常	3
		more	センサ故障	ガイド表示異常	3
シフトレバー	入力値 = {P, R, N, D, 2}	no	センサ故障 センサ結線故障	表示切替不可	2
		reverse	センサ故障	表示切替異常	3
		other than	センサ故障	表示切替異常	3
フレームバッファ	出力データ周期 = 30(fps)	no	カメラ故障 アプリ停止	画像停止	4
		less	カメラ異常 アプリ異常	コマ落ち	3
		more	カメラ異常 アプリ異常		1
	出力値	part of	カメラ異常 アプリ異常	部分的表示	3
		reverse	カメラ異常 アプリ異常	画像ノイズ	3
	遅延時間 < 30(ms)	late	アプリ異常	表示遅れ	4

3.3 監視条件の導出

この HAZOP 分析結果 (表 3) をもとにして，監視ポリシーの導出を行う．監視ポリシーは監視条件と回復手順から構成される．

監視条件は仕様とガイドワードから導出できる．HAZOP では仕様外の動作についてガイドワードを適用することで分析を行うものであるので，HAZOP 分析結果の行ごとに監視条件を設定することができる．

回復手順は別途導出する必要があるが，HAZOP 分析による原因欄を利用することが可能である．結線故障・センサ故障のようなハードウェア的な原因である場合はソフトウェアによる回復は不可能である場合も考えられるが，一時故障でリセットによって回復できる場合のことを想定して入力センサ・カメラが原因となっている場合にはそのコンポーネントのリセットを行うことで回復手順とする．

このようにして導出した監視ポリシーが**表 4** である．

監視ポリシーを定めるためには，仕様の曖昧性を解消する必要がある．具体的に表 4 では，カメラ入力周期およびフレームバッファ出力周期についてガイドワード less の適用条件として fps<20 としている部分，および，ガイドワード part of/reverse の適用条件として CRC によるエラー検出を仮定している部分などがそれにあたる．これらの条件値は，仕様からどこまで逸脱した場合に異常とみなすかの閾値として設計時に検討を行い，定義する必要がある．

表 4 バックガイドモニタ監視ポリシー

監視対象	ガイドワード	監視条件	継続時間 (s)	回復手順	危険度
カメラ	no	タイムアウト	1	カメラリセット	4
	less	fps < 20	3	カメラリセット	3
	more	fps > 30	3	カメラリセット	3
	part of	CRC エラー	1	カメラリセット	3
	reverse	CRC エラー	1	カメラリセット	3
	other than	-	-	-	-
ステアリング	no	タイムアウト	60	センサリセット	2
	less	入力値 < -100	即時	センサリセット	3
	more	入力値 > 100	即時	センサリセット	3
シフトレバー	no	タイムアウト	60	センサリセット	2
	reverse	入力値 = 列挙値以外	即時	センサリセット	3
	other than	入力値 = ステアリング値	1	アプリ停止	3
フレームバッファ	no	タイムアウト	1	カメラリセット・アプリ再起動	4
	less	fps < 20	3	カメラリセット・アプリ再起動	3
	more	fps > 30	3	カメラリセット・アプリ再起動	1
	part of	CRC エラー	1	カメラリセット・アプリ再起動	3
	reverse	CRC エラー	1	カメラリセット・アプリ再起動	3
	late	出力時刻 - カメラ入力時刻 > 30ms	即時	アプリ再起動	4

このような手順によって HAZOP を用いて監視ポリシーを導出することの利点は，監視条件ごとに危険度も導出可能であるということにある．一般にソフトウェアによる動作監視においては，監視条件を増やすことで安全性を高めることが可能であるが，その一方で負荷が上昇するため無制限に監視条件を増やすことはできないというトレードオフの関係が存在する．このとき，危険度の高い監視条件を優先的に監視し，危険度の低い監視条件は監視対象から外すことによって，一定限度の負荷で安全性を高めることができると考えられる．

4. 実　　装

われわれは本手法を制御系システムおよび制御系システムと連携して動作する情報系システムへ適用することを目標としている．本研究では後者の組込み系システムと連携して動作する情報系システムへの適用例として，Unix 系 OS 上での実装を行った．本実装の構成図を**図 3** に示す．

2 章で述べたソフトウェア監視機構に当てはめると，この実装は**表 5** に示す要素で構成されていると対応付けることができる．

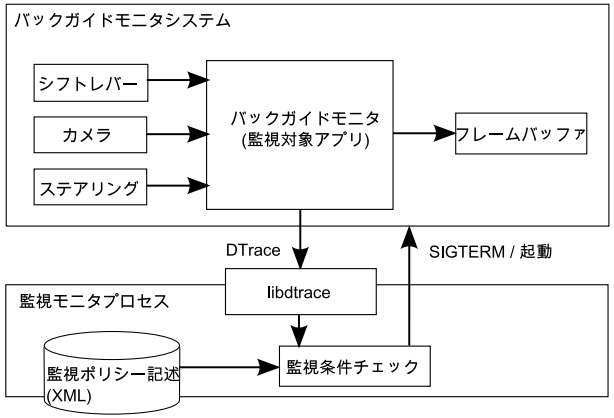

図 3 ソフトウェア監視機構実装

表 5 実装

監視システム構成要素		実装
監視対象アプリケーション		ユーザプロセス
監視モニタ		ユーザプロセス
	監視ポリシー導出手法	HAZOP
	監視情報プロービング	DTrace
	異常検出	監視ポリシーによる条件チェック
	回復手法	シグナル

監視対象アプリケーションと監視モニタは，それぞれをユーザプロセスとして記述した．また，監視対象アプリケーションの入出力対象として，外部デバイスのエミュレータをそれぞれ別プロセスとして実装し，TCP/IP によって入出力を行うものとした．

4.1 DTrace による監視情報取得

監視情報のプロービング手法には，Sun Microsystems によって開発された DTrace[5] を利用している．DTrace は，Solaris，FreeBSD，MacOSX などの OS に実装されている実行時トレース取得機構であり，OS カーネルおよびユーザプロセスの各種情報を取得し，D 言語で記述されたアクションを実行することが可能である．DTrace は OS カーネル内に組み込まれたサブシステムであり，通常は参照することが不可能な他プロセスの動作情報に対してプローブを設定できる点で，監視情報プロービングの手段として有用であると考えている．

本研究では，監視対象アプリケーションの入出力を監視するため，ソケット入出力に関するシステムコール呼び出しに対するアクションを記述した．また，libdtrace[6] を用いて情報を取得するコンシューマとして監視モニタプロセスを実装した．この監視モニタプロセスで対応しているガイドワードについては no, less, more, other than, early, late である。今回の実装では表 2 の適用対象における構造値の部分が未実装であり，part of, reverse については実装していない．

DTrace を用いてプロセスの動作を取得するためには，D 言語によるスクリプトを記述して行う．今回実装した周期監視については，リスト 1 のようなアクションを記述している．syscall プロバイダを用いて read/write それぞれのシステムコールの呼び出し回数をファイルディスクリプタごとにカウントして集積体に記録し，profile プロバイダを利用してその数を監視プロセスで処理するために 1 秒に 1 回出力している．D 言語にはループなどの構造を持たないが，ここで利用している count() 関数のような集積関数を効率よく実行できることが最大の特徴である．

リスト 1 interval.d

```
1  syscall::write:entry
2  /pid == $target && trace_count_rw/
3  {
4      @write_count[arg0] = count();
5  }
6  syscall::read:entry
7  /pid == $target && trace_count_rw/
8  {
9      @read_count[arg0] = count();
10 }
11 profile::tick-1s
12 /trace_count_rw/
13 {
14     printa("MSG: count_rw write sock=%d, count=%@d\n", @write_count);
15     printa("MSG: count_rw read sock=%d, count=%@d\n", @read_count);
16     clear(@write_count);
17     clear(@read_count);
18 }
```

同様に，範囲値/列挙値の監視については，リスト 2 のようなアクションを記述している．syscall プロバイダによって read システムコールの呼び出しから戻る時点で読み込みバッファに書き込まれている値を取得し，その値を監視プロセスへ出力している．読み込みバッファは監視対象プロセスのメモリ空間上に存在しているが，DTrace の copyin() 関数によってその内容へアクセスすることが可能となる．

Vol.2009-EMB-14 No.1
2009/7/24

リスト 2　value.d

```
1  self int read_sock;
2  self int read_buf;
3
4  syscall::read:entry
5  /pid == $target/
6  {
7      self->read_sock = arg0;
8      self->read_buf = arg1;
9  }
10 syscall::read:return
11 /pid == $target && 0 < arg0 && trace_value /
12 {
13     printf("MSG: value read sock=%d, buffer=%d\n",
14      self->read_sock,
15      *((char*)copyin(self->read_buf, 1)));
16 }
```

4.2　監視条件判定と回復

コンシューマプロセスでは libdtrace を用いることで D 言語スクリプトからの出力文字列を取得し，仕様値からの逸脱をチェックすることが可能である．DTrace を用いる場合はいったん文字列を介する必要があることが欠点であり，D 言語スクリプトから直接コンシューマプロセスへメッセージ通信を行うよう拡張することでさらに効率的な監視情報の取得が可能となる．

異常からの回復手法については，SIGTERM シグナルを対象プロセスへ送出していったんプロセスを終了し，再起動することで行うものとした．センサのリセットについても同様にエミュレータプロセスへのシグナル送出と再起動によって実行している．

5. 評　　価

本実装の評価のため，危険度ごとに監視ポリシーを設定して実行したときのプロセス負荷率とメモリ使用量を測定した結果を**図 4** に示す．

今回の実装では Intel Core2Duo + VMware Server 上で動作させているため，組込み機器への適用へ向けて，相対的な実行時間比とメモリ使用量の変化について評価を行うものとする．

グラフ横軸は no: 監視機構なし, null: 監視機構組込み・監視項目なし, noval: 監視機構での値チェックなし・周期監視のみ, all: 監視機構での値チェック・周期監視組込みを表しており，右に行くほど監視機能が多数組み込まれていることを示す．監視ポリシー表 4 には，all が全ての監視条件を適用していることを，noval が危険度 4 以上の監視条件を適用していることに対応する．グラフ左縦軸は，user 時間/sys 時間 (単位:%)，右縦軸は使用メモリ量 mem(単位:KB) を表している．

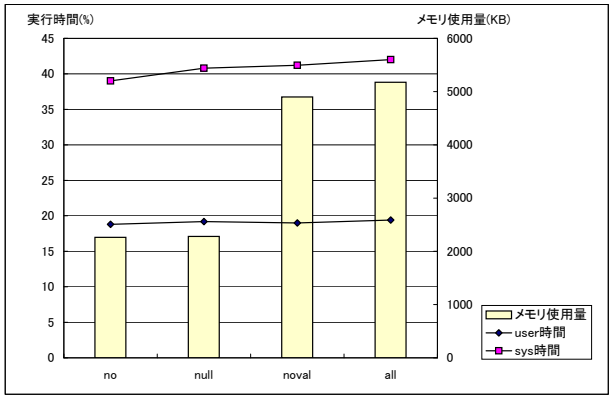

図 4　測定結果

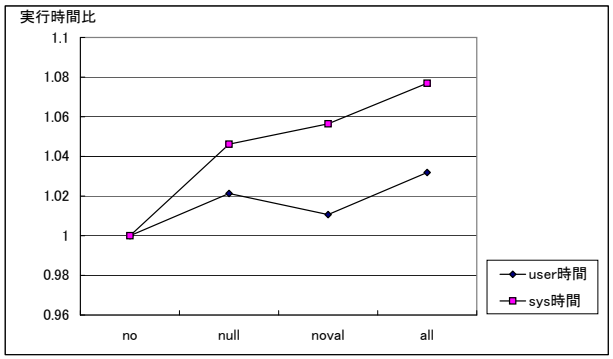

図 5　監視機構なしの場合との実行時間比

負荷についてさらに分析したグラフを**図 5** に示す．このグラフでは監視機構なしのときの実行時間 (user,sys) を 1 とした実行時間比を示している．

全ての条件について監視機構を組み込んだ場合で実行時間の増加比は 8%程度であり，危険度 4 の条件だけを適用することで負荷の増加を 6%以下に抑えることが可能となっている．

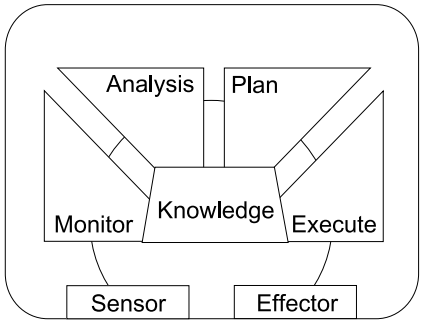

図 6 MAPE-K 参照モデル

これらのグラフより，危険度によって実行時に適用する監視条件を絞り込むことによって負荷率を調整することが可能であり，安全性と負荷についてのトレードオフについて調整することが可能であることが示された．一方，必要メモリ量については，現在の実装では周期監視を行うだけで 3MB 程度増加している．これは，DTrace の処理バッファを固定長で確保しているためであるが，組込み用途を考えるとさらに削減・調整することが必要であり，今後の課題として考えている．

6. 関 連 研 究

本研究のような監視機構の例としては，IBM によって**図 6** のような MAPE-K 参照モデル[7] が提案されている．これは主にサーバを中心としたエンタープライズシステムへの適用を目的としたモデルであるが，同じく IBM によって組込みシステムへの適用[8] も提案されている．

MAPE-K 参照モデルでの監視機構は Monitor,Analysis,Plan,Execution,Knowledge の 5 つの要素で構成される．**表 6** に，本研究における MAPE-K 参照モデルとの対応関係を示す．本研究では Knowledge,Analysis の分析方法として HAZOP を，Monitor 機構として DTrace を用いることで，組込みシステムへの監視機構の適用について検討したものだととらえることができる．

Plan 部分については，本研究では未検討である．今回の実装では監視条件に合致した場合に単純にシグナルを送信しているが，複数の監視条件が検出された場合の動作について今後検討が必要であり，この部分が Plan として実装されていく必要があると考えている．

表 6 MAPE-K 参照モデルと本実装の対応

MAPE-K 参照モデル	本研究における実装	
Monitor	監視情報プロービング手法	DTrace
Analysis	異常検出	監視ポリシーによる条件チェック
Plan	-	-
Execution	回復手法	シグナル
Knowledge	監視ポリシー導出手法	HAZOP

装置やシステムの外部からの攻撃を監視する異常検出 (Anomaly detection) システムや侵入検知システム (Intrusion Detection System) も監視機構の一つの形態と考えられる．異常検出の手法としては，大きく分けてブラックリスト方式とホワイトリスト方式と呼ばれる 2 種類の方式がある．ブラックリスト方式は異常検出条件について記述する方式であり，ホワイトリスト方式は正常動作について記述してそれを逸脱した場合に異常として扱うものである．

ブラックリスト方式の研究例としては，異常を形式的に記述する方法[9]，CPU やネットワークの利用状態を入力しマルコフモデルなどを利用して事象を解析する方法[10] などがある．汎用のネットワーク機器やサーバに対する侵入検知システムにおいてはブラックリストを作成することが困難であるためにホワイトリスト方式が着目されているが，ブラックリスト方式と同様に正常動作について形式的に記述する方法などもあるが，学習フェイズに正常動作時の情報を蓄積しておいてそこからの逸脱を検出する方式[11] が多く研究されている．

われわれの試験実装している方式はブラックリスト方式にあてはまるが，本研究では対象が制御系システムであり，HAZOP による監視条件検出という仕組みは，仕様書をホワイトリストとして，あらかじめ設計時に仕様からの逸脱について分析することでブラックリスト作成を容易にしている．また，通常のホワイトリスト方式では正常時から逸脱していることを検出した場合にその結果としてどのような危険をもたらすものであるかは判別できないが，われわれの方式によれば，設計時に分析を行うことで危険度を割り当て，実行時に異常を検出した際に，その危険度に応じた回復を行うことができるという利点があると考えている．

7. お わ り に

本研究では，ソフトウェア監視機構の監視条件の導出手法として HAZOP を利用する方法について説明した．また，実行環境におけるソフトウェア監視機構を DTrace を用いることによって実装し，監視条件を増加させるとき負荷が増大し，負荷低減のために監視条件を

絞り込む必要があるという制約のもとで，HAZOP 分析によって得られる危険度を用いることで負荷を調整できる可能性について示した．

今後の方針としては，対応するガイドワードの追加，CRC などの基本的な異常検知機構の組込みを行ったうえで，複合した異常への対応のために MAPE-K モデルの Plan 部分の検討を進めていくことが必要である．また，ミドルウェア化を進め，RTOS への組込みを行うことで，情報系システムのみならず制御系システムへの適用が可能であると想定している．

謝辞 本研究を進めるに当たり，貴重なご意見をいただいたトヨタ自動車株式会社 BR 制御ソフトウェア開発室の方々をはじめ，ご協力くださった名古屋大学大学院情報科学研究科附属組込みシステム研究センター車載マルチメディアシステム向け OS プロジェクトのメンバー各位に厚く御礼申し上げます．

参 考 文 献

1) 井上洋一，平尾裕司，蓬原弘一，川池　嚢，向殿政男：制御システムの安全―ISO13849-1(JIS B9705-1)、IEC60204-1(JIS B9960-1)、IEC61508(JIS C0508) (安全の国際規格)，日本規格協会 (2007).
2) IEC 61882: Hazard and operability studies (HAZOP studies)- Application Guide (2001).
3) Redmill, F., Chudleigh, M. and Richard, J.C.: *System Safety: HAZOP and Software HAZOP*, Wiley-Blackwell (1999).
4) IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems (2005).
5) McDougall, R., Mauro, J. and Gregg, B.: *Solaris Performance and Tools : DTrace and Mdb Techniques for Solaris 10 and OpenSolaris*, Prentice Hall (2006).
6) Sun Microsystems, Inc.: libdtrace(3LIB), http://docs.sun.com/app/docs/doc/816-5173/6mbb8adt2.
7) IBM: An Architectural Blueprint for Autonomic Computing, http://www.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf (2006).
8) 秋山一人，鈴木康裕，津村直史，西村康孝：組み込みデバイスのためのオートノミック・コンピューティング・アーキテクチャー，PROVISION 58, IBM (2008).
9) Meadows, C.: A Formal Framework and Evaluation Method for Network Denial of Service, *Processings of the 1999 IEEE Computer Society Foundations Workshop*, pp.4–13 (1999).
10) Sugaya, M., Ohno, Y., vander Zee, A. and Nakajima, T.: A Lightweight Anomaly Detection System for Information Appliances, *Processings of IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pp.257–266 (2009).
11) Waizumi, Y., Kudo, D., Kato, N. and Nemoto, Y.: A New Network Anomaly Detection Technique Based on Per-Flow and Per-Service Statistics, *Computational Intelligence and Security*, Splinger Verlag., pp.252–259 (2005).