社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－OS－97 (14)
2004／8／1

自律的システム管理ソフトウェアの設計

岡家 豊[†] 木村 かず子[†] 石川 裕[††]

IPMI (Intelligent Platform Management Interface) 規格は、ボード上の温度、電圧、冷却ファンなどの機器状態を監視するハードウェアおよびそれらの機器状態をソフトウェア側で利用するためのインタフェース仕様である。我々は、当規格に基づき信頼性、可用性、保守性、管理性を備えたいわゆるディペンダブルシステムを実現するための管理ソフトウェアを開発している。本管理ソフトウェアは、IPMI で提供される管理機能に加え、管理専用のスクリプト言語を提供することで、ユーザが表示内容や表示方法をカスタマイズする機能が提供される。さらに、システム管理者がサーバ監視のためのルールを記述することにより、自律的なサーバ運用を可能とする機能が提供される。管理ソフトウェアの開発と共に IPMI 規格に基づく機器故障シミュレータも開発している。本シミュレータを使用することにより、管理ソフトウェアのカスタマイズやルール記述の動作検証が可能となる。

The Design of Autonomous System Management Software

YUTAKA OKAIE,[†] KAZUKO KIMURA[†] and YUTAKA ISHIKAWA[††]

IPMI (Intelligent Platform Management Interface specification) is the specification of the interface between software and hardware that observes the state of devices, such as temperature, voltage, cooling fan, and so on. Using the IPMI protocol, a maintenance system has been developed to realize a reliable, available, and manageable system so-called a *dependable system*. In addition to the management functions provided by IPMI, a new script language is provided so that the users may customize the contents to be displayed, and describe management rules so that the managed system runs autonomously. A failure simulator based on IPMI has been also developed as a testing environment for the maintenance system. This software enables us to check the functions.

1. はじめに

企業における基幹系システムを中心に高い信頼性、可用性、保守性、管理性を備えたシステムを実現するための管理保守系システムソフトウェアが開発されてきている。管理保守系システムソフトウェアには、故障診断ツール、監視・通報ツールなどが含まれる。例えば、システム障害が生じると監視通報ツールが自動的に遠隔地にある監視センタに通報する。監視センタでは遠隔操作によりシステム障害が生じたシステムの保守プロセッサを使用して故障診断ツールを動かし、障害を特定する。

従来、企業における基幹系システムで使われているサーバコンピュータでは、ハードウェアメーカ毎に独自の監視ハードウェア上で管理・保守ツールが開発されてきた。1998 年以来、IPMI イニシアティブ (Intel、DELL、Hewlett-Packard および NEC の 4 社) によって IPMI (Intelligent Platform Management Interface) と呼ばれる監視システムの規格化が行なわれている。本規格は、サーバの温度、電圧、冷却ファン、電源などを監視するハードウェア仕様を標準化している。最近の x86 系 CPU を搭載したサーバ系のボードには、IPMI 規格に準拠した監視プロセッサが搭載されている。

我々は、IPMI 規格に準拠したサーバシステム上で稼働する管理保守系システムソフトウェアを Linux カーネル上で開発している[4]。本管理・保守システムは、被監視サーバ上の障害を検知すると監視センタに通報する機能だけでなく、故障の度合により自動的に冗長部品に切り替える機能やスタンバイコンピュータへのサービス移行機能などを提供する。

このようなシステムソフトウェアの開発および運用では、開発あるいはカスタマイズ時に、機器故障に対応する自動運転機能が仕様通りに動作しているか確認するテスト環境が必須である。そこで、我々は、管理保守システムソフトウェアの開発と同時に機器故障を模擬するシミュレータも開発している。故障シミュレータは、IPMI 規格で定義されているセンサ情報やベンダ固有情報に基づいて人工的に情報を生成するものであり、Linux カーネル上の一プロセスとして稼働する User Mode Linux (UML)[5] のカーネルモジュールとして実現される。

本稿では、第 2 節で、IPMI 規格の概要を述べた後、第 3 節で自律システム管理ソフトウェアの設計につい

† NEC ソフト株式会社
†† 東京大学

て述べる。第4節で IPMI に基づいた故障シミュレータの実現方法について述べる。第5節で関連技術について述べる。

2. IPMI 1.5 規格

IPMI 規格[1] はサーバの温度、電圧、冷却ファン、電源などを監視する機器の仕様を定めたものである。各機器には自動監視、ロギング、自動復帰といったマネジメント機能がハードウェアもしくはファームウェアに直接実装されている。本規格では、これらのハードウェアとソフトウェア間のインターフェース仕様についても定義されている。

2.1 IPMI インタフェース

IPMI 規格に基づくハードウェアの構成は図1のようになっている。ボード上には **Baseboard Management Controller (BMC)** と呼ばれるコントローラがあり、ハードウェアと管理ソフトウェア間の標準インタフェースを提供する。

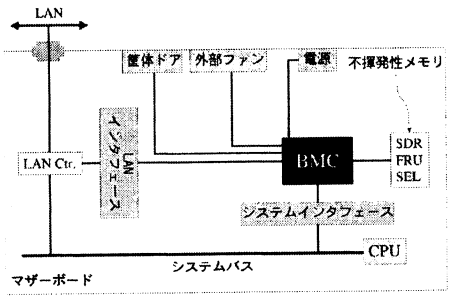

図1 IPMI のハードウェア構成

管理ソフトウェアは、当規格で定められたインタフェースを介し、BMC とデータをやりとりすることで、監視対象となるサーバ上のハードウェア状況を管理する。管理ソフトウェアが BMC にアクセスするインタフェースとして LAN インタフェースおよびコンソールインタフェースが提供されている。

コンソールインタフェースを利用しての BMC へのアクセスは、IPMI 規格で定められたシステムインタフェースと呼ばれるデバイスを経由して行われる。そのため、管理ソフトウェアはシステムインタフェースのドライバを介して BMC へのアクセスを行う。このドライバの実装は様々なベンダや、オープンソースのプロジェクト[10] によって提供されているが、これらを総称して以下 IPMI ドライバと呼ぶ。一方、LAN インタフェースの場合は、IPMI ドライバを介さず UDP を用いて BMC に直接アクセスを行うよう定められている。

2.2 BMC が提供する管理機能

BMC が IPMI インタフェースを介して管理ソフトウェアに提供する機能について述べる。

表1 センサタイプの概要

タイプ	概要
温度	CPU 温度
電圧	ボード上の 5V 系、12V 系の電圧
ファン	CPU ファンの回転数
筐体イントリュージョン	筐体のドアが開閉、 LAN ケーブルの装脱着
プロセッサ	熱暴走等の異常
メモリ	ECC エラー
電源ボタン	電源、リセット等のボタン押下
LAN	NIC 異常
バッテリ	バッテリ残量

2.2.1 センサ情報の管理機能

筐体内に存在するセンサ情報は、図1の不揮発メモリ上に保持されている。管理ソフトウェアは BMC を介して不揮発メモリに保持されているデータを取り出すことができる。当規格で定められているセンサ情報には以下のものがある。

- センサの種類：ボードの温度、電圧、ファン回転数など
- センサの読み取り値の種類：連続値 (スレシュホールドを持つもの)、離散値 (複数の異常状態を監視しているもの、メーカ独自仕様のもの)
- スレシュホールド値：上回る (下回る) とハードウェア異常が発生したとみなされる値 (センサの読み取り値が連続値である場合のみ)
- 発生し得るイベントの種類：発生したハードウェア異常の種類。例えば CPU 熱暴走、FAN 回転数低下、メモリの ECC エラーなど
- センサ ID：メーカが独自に付加するデータ
- 読み取り値換算に必要な値：変換式、係数など

具体的な例として、1.2 V の電圧センサの情報は以下のようになる。

- センサの種類：ボード電圧
- センサの読み取り値の種類：連続値
- スレシュホールド上限値：1.31
- スレシュホールド下限値：1.08
- 発生し得るイベント：電圧値がスレシュホールド値を下回る
- 単位：Volt
- センサ ID：Baseboard

IPMI 規格で定められているセンサのタイプは全部で 41 種類ある。ここでは、主だったタイプの概要を表1に示す。これらのセンサ情報は **Sensor Data Record (SDR)** として不揮発メモリ上に保持されている。

2.2.2 イベントのロギング

BMC は SDR に保持されているセンサ情報に基づいて各機器の監視を行っている。ある機器に異常 (e.g. CPU 温度がスレシュホールドを越えた、電源が落ちている、など) が発生した時、その旨をロギングする

機能を持っている。この時、発生した異常は **イベント** と呼ばれ、そのログは **System Event Log (SEL)** として不揮発メモリ上に保存される。SEL として書き込まれる情報には、以下のものが含まれる。

- イベント発生時刻
- イベント内容：以上が発生したセンサの種類、識別子、および、発生したイベントの種類
- イベントの重要度：情報、状態回復、異常状態へ変化など

2.2.3 イベント通知機能

故障などのイベントが発生した時に、その旨を **Alert** として通知する機能がある。アラート通知方法の一つに LAN を経由するものが定められている。

3. 自律システム管理ソフトウェアの設計

本節では、まず、自律システム管理ソフトウェアを設計する上で最も重要である機器の状態変化と、それに対する扱いについて述べる。これを踏まえた上で、提案システムの設計について述べる。

3.1 機器の状態変化とその扱いについて

機器の状態は、ソフトウェア要因による状態と機器を監視しているセンサ情報に基づく状態の 2 種類がある。ソフトウェア要因の例としては、管理者の shutdown コマンド実行によるシステム停止が挙げられる。センサ情報に基づく状態は、例えば、IPMI 規格に基づいた BMC から得られる情報であり、CPU 温度の高温閾値を越えているような状態が挙げられる。

DMTF (Distributed Management Task Force) が規格化している CIM (Common Information Model) では、機器の状態変化に伴い発生する現象をインディケーション、イベントの 2 種類に分けている [15]。イベントは現象 (phenomenon of interest) が生じたことであり、インディケーションはイベント (event of interest) 検出の記録であると定義している。例えば、メモリの ECC 検出（イベント）はインディケーションとして発生する。CIM はインディケーションのみを扱っている。

インディケーションの発生を受けて、機器の状態が変化するという機器モデルを考えるのが自然であると考える。例えば、上記 ECC インディケーションが多数発生した場合でも、同一メモリバンクからインディケーションであった場合には、一つのイベントとして扱い、そのメモリバンクがエラー状態に遷移し、その状態に応じたアクションを起こす。言い替えると、イベントによって状態は遷移し、状態に応じたアクションを起こし、機器を制御する必要がある。これら、イベント、状態、状態に基づくアクションを定義できる機能が必要である。

3.2 *Tenjin* の概要

我々が開発している保守管理ソフトウェア（開発コード名 *Tenjin*）は、図 2 に示すようなモジュールから構成され、次のような機能を提供する。

- WEB サーバエンジン
  サーバサイドプログラミングにより、機器の表示系、システム運用ポリシの設定などを行なう。新たに、ATML (A Temporal Markup Language) と呼ばれるタグ体系を定義し、ATML で記述される。
- 監視エンジン
  Tenjin 監視スクリプト言語によって記述された監視プログラムを実行する。*Tenjin* 監視スクリプト言語と ATML は一つの言語として設計されている。すなわち、ATML のタグ体系に対応して *Tenjin* 監視スクリプト言語構文が定義されていて、その逆も成り立っている。
- データベースエンジン
  DMTF が規格化している CIM に基づいたデータベースエンジンであり、機器情報および機器の状態が格納される。
- プロバイダ
  プロバイダは、機器情報を取得し、データベースエンジンに情報を登録、あるいは、機器からの Alert メッセージを受け取って、監視エンジンにインディケーション (第 3.4.3 節参照) を発生させる機能を提供する。現在、プロバイダとして、IPMI だけを想定しているが、SNMP[17] 用プロバイダも実現していく。

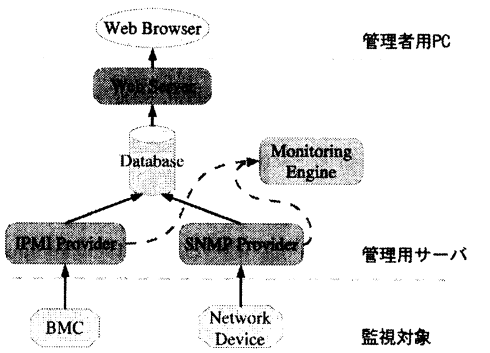

図 2 *Tenjin* システムの概要

3.3 データベース定義

CIM に基づいて、監視する機器データベースを構築する。CIM は、MOF (Managed Object Format) によって定義される。CIM V2.8 で定義されている MOF ファイルには、1184 個のクラスが定義されている。

例えば、以下は、クラスタ CIM_Class の定義例である。class キーワードの前にある [と] で囲まれている式は、Qualifier である。クラスの Qualifier のインスタンスとして、Version ならびに Description

```
print("
<H3>MyCluster 情報</H3>
<TABLE>
<TR>
    <TH>管理者</TH>
    <TH>連絡先</TH>
</TR>
");
foreach(query("select * from CIM_Cluster
       where Name='MyCluster'")) {
   print("<TR><TD>%s</TD>",
         eval("PrimaryOwnerName"));
   print("<TD>%s</TD></TR>",
         eval("PrimaryOwnerContact"));
}
print("</TABLE>");
```

図 3 foreach 構文を用いた query 関数使用例

が定義され、CIM_Class が V2.6 で定義されたこと、ならびに本クラスの説明が記されている。本クラスのスーパクラスは CIM_ComputerSystem である。本クラスは、参照可能なプロパティとして、Interconnect および InterconnectAddress が定義されている。

```
[Version ( "2.6.0" ),
   Description (".....")]
class CIM_Cluster : CIM_ComputerSystem {
   [Description (".......")]
 string Interconnect;
   [Description (".......")]
 string InterconnectAddress;
  ...

};
```

このようなオブジェクト定義から SQL のテーブルに変更するコンパイラを実現する。MOF で定義されている Qualifier などの属性も SQL テーブルで管理する。

3.4 監視スクリプト

監視スクリプトと次節で述べる ATML (A Temporal Markup Language) は一つの言語ファミリとして設計されている。監視スクリプトで定義される構文に対応する ATML タグが存在し、監視スクリプトで定義されたプログラムは ATML タグでも記述が可能である。

3.4.1 データベース検索

データベース検索のために query 関数を導入する。本関数の戻り値は検索結果のリストである。図 3 では、CIM_Cluster テーブル中、Name カラムが"MyCluster" であるエントリを検索している。foreach ブロック内では、eval 関数を用いて選択されたエントリのカラムを参照することが出来る。図 3 では、print 関数を使うことにより HTML ドキュメントを生成している。

3.4.2 時間定数と関数

数字の後に Y、M、d、h、m、s を付加することにより、年、月、日、時、分、秒を表現する。例えば、2004Y、11M、20d、8h、20m、30s は、それぞれ、2004 年、11 月、20 日、8 時、20 分、30 秒を表現している。

3.4.3 インディケーション、イベント、状態定義

監視機器から上がってくるイベントをインディケーションとして扱う。インディケーションを受け取って、イベントを生成する機構、およびイベントを受け取って処理する機構を提供する。

インディケーションおよびイベントが発生していると真を返す onIndication、onEvent 述語をそれぞれ導入する。イベント生成のための関数として raiseEvent 関数を、状態遷移させるための関数として、changeState 関数を提供する。

述語が成立した時に起動するアクションを定義する構文として rule 構文を導入する。rule 構文は、ルールが真の時に実行されるアクション部を伴う。下記の例では、IPMI プロトコルで生成された温度に関するインディケーションが発生すると、{ と } で囲まれたアクション部が実行される。ここでは、raiseEven 関数の第一引数に this 変数を指定している。this 変数は、述語が成立した時の CIM_ThresholdIndication オブジェクトを指す。

```
rule(onIndication('Tempurature',
                  SENSOR_IPMI)) {
   /* CIM_ThresholdIndication オブジェクトの
    * プロパティがアクセスできる */
   raiseEvent(this, SENSOR_IPMI)
}
```

以下の例では、'HighTempurature' イベントが発生すると、アクション部が実行される。changeState 関数の第二引数の AlertingManagedElement は、CIM_ThresholdIndication オブジェクトのプロパティで、イベントを発生した機器オブジェクトを指している。

```
rule(onEvent('HighTempurature',
             SENSOR_IPMI)) {
/* CIM_ThresholdIndication がアクセスできる */
   changeState('StateHighTempurature',
               AlertingManagedElement);
}
```

3.4.4 sometime および always 時制述語

ある状態が続いてた時に真となる述語として、sometime と always を導入する。以下の sometime 述語使用例では、過去 1 時間に渡って 50 % の間、'HighTempurature' 状態が続いていたならば、アクション部分が実行される。

```
rule(sometime('HighTempurature',
```

```
                past(1h), 0.5)) {
}
```

always 述語は、sometime 述語の第三引数が 1.0 の時と同じで、以下のような記述となる。

```
rule(always('HighTempurature',
            past(1h))) {
}
```

3.5 ATML

Zope[16] が提供する DTML (Dynamic Template Markup Language) のようなタグとして ATML (A Temporal Markup Language) を導入する。

3.5.1 名前空間

変数の検索順序は以下の通りである。

(1) ATML タグによって動的に生成される名前空間。ATML タグがネストすることにより名前空間もネストする。

(2) 呼び出された時の PATH 上に存在するファイル名。あるいは、その PATH から辿れる全ての親ディレクトリに存在するファイル名。

(3) request で定義されている名前。

3.5.2 基本 ATML

以下、プログラミングに必要な基本タグを導入する。

- `<atml-let>`
  変数を定義するタグ。

- `<atml-eval>`
  変数あるいは式の評価により得られるテキストを挿入する。例えば、Foo というファイルに
  `<H2>Hello World !</H2>`
  という一文が格納されていた場合、以下のタグは上記の一文に置き換わる。
  `<atml-eval name="Foo">`

- `<atml-if> <atml-else> </atml-if>`
  条件文を実現するタグである。例えば、Value というファイルに 100 という値が入っていた時、以下の文が評価されると、条件式が満たされるので、`<P>Value is grater than 10</P>` が表示される。

```
<atml-if expr="Value > 10">
    <P>Value is grater than 10</P>
<atml-else>
    <P>Value is no more than 10</P>
</atml-if>
```

- `<atml-call>`
  関数を呼び出すタグである。

- `<atml-return>`
  関数の戻り値を指定するタグである。

- `<atml-try> <dtml-except> </dtml-try>`
  例外ブロックを実現するタグである。

- `<atml-foreach> </dtml-foreach>`
  イテレーションを実現するタグである。

3.5.3 データベース検索

データベース検索は `<atml-foreach>` を用いて実現される。例えば、図 4 の左のプログラムは、図 3 の例を ATML で記述した例であり、MyCluster という名前を持つクラスタの管理者および連絡先を表示する HTML を生成する。

3.5.4 ルール

第 3.4.3 節、および第 3.4.4 節で導入したルール構文に対応する ATML タグは `<atml-rule>` である。以下に例を示す。

```
<atml-rule
    expr="sometime('HighTempurature',
                   past(1h), 0.5)">
<atml-action script="bash"
             args=AlertingManagedElement>
  shutdown $1
</atml-action>
</atml-rule>
```

3.5.5 グラフ表示

グラフを表示するためのタグとして `<atml-graph>` タグを導入する。以下の 3 つのタグの例はそれぞれ、折れ線グラフ、棒グラフおよびタコメータを表示する。

```
<atml-graph type=linegraph file=File>
<atml-graph type=bargraph
            data="(1, 2) (2, 4)">
<atml-graph type=tachometer min=-30 max=200
            data="70">
```

4. IPMI に基づく故障シミュレータ

本節で述べる故障シミュレータは、IPMI 規格に基づいた管理ソフトウェアの開発過程で、その動作検証の際、実際にハードウェア障害を発生させる代わりに、ソフトウェア的にハードウェア障害を発生させるためのツールである。このようなソフトウェアが存在しなかった従来、管理ソフトウェアの動作検証には、実際にハードウェアの異常を発生させたり、障害を模擬的に発生させるために各ハードウェアに関する専門的な知識が必要であった。

本故障シミュレータは IPMI 規格に基づいて作成されており、ソフトウェア的に BMC の動作をエミュレーションすることで、ハードウェア障害をシミュレーションするソフトウェアである。これにより、ハードウェア障害を発生させての動作検証にかかる工数、およびコストを大幅に削減することが可能である。IPMI 規格に準拠した特別なハードウェアなしに、本ソフトウェアを使用することができる。すなわち IPMI 規格

```
<H3>MyCluster 情報</H3>
<TABLE>
<TR>
    <TH>管理者</TH>
    <TH>連絡先</TH>
</TR>
<atml-foreach
    query="select * from CIM_Cluster
           where Name='MyCluster'">
<TR>
    <TD><atml-eval
         name="PrimaryOwnerName"></TD>
    <TD><amtl-eval
         name="PrimaryOwnerContact"></TD>
</TR>
</atml-foreach>
</TABLE>
```

ATML 記述例

```
<H3>MyCluster 情報</H3>
<TABLE>
<TR>
    <TH>管理者</TH>
    <TH>連絡先</TH>
</TR>
<TR>
    <TD>岡家 豊</TD>
    <TD>okaie@is.s.u-tokyo.ac.jp</TD>
</TR>
</TABLE>
```

実行結果例

図 4 atml-query タグ使用例

に準拠した高価なサーバを用意することなく、IPMI 規格に基いて作成された管理ソフトウェアの開発支援を行うための機能が提供される。

以下の小節では、前節で提案した管理ソフトウェアの動作検証を行う環境として、本故障シミュレータに必要とされる検証支援技術について述べる。その後に、本故障シミュレータの実現方法について述べる。

4.1 検証支援技術

IPMI 規格に基づく本故障シミュレータは BMC の動作をエミュレーションすることで実現される。本故障シミュレータを用いて動作検証を行う場合、管理ソフトウェアは、ソフトウェア的に実現された BMC にアクセスすることで、ハードウェアの様々な状況下における動作検証を行うことができる。前節で提案した管理ソフトウェアの動作検証を行う環境として本故障シミュレータを捉えた場合、BMC のエミュレーション機能に加え、さらに以下の機能が必要である。

4.1.1 管理ソフトウェア開発支援機能

信頼性の高いソフトウェアを開発するためには、バグが発生した状態を再現できる環境を構築し、発見したバグを取り除くことが必要である。また、状態を再現する機能だけでなく、どのような状態にある時にバグが発生するのか特定できる機能も必要である。

本故障シミュレータのようなハードウェア障害を発生させるシミュレータの場合、どのようなハードウェア状態に遷移した時にバグが発生しているのか、その他の複合的な要因がバグの原因になっているのかを特定する手がかりとして、ハードウェア状態の変化の時系列データをロギングする機能が必要である。この機能により、バグの原因となるハードウェア状態を推定することができる。その状態を再現する機能の実装として、ハードウェア障害の種類、発生順序、およびタイミングなどの障害発生状態を詳細に記述できるものを提供すべきである。

4.1.2 サーバ環境のシミュレーション機能

近年、クラスタやグリッドなど、複数のサーバが協調してシステムが構成されるようになってきた。そのような環境をターゲットとして作成された管理ソフトウェアの動作検証には複数台のサーバ環境をシミュレーションする機能が必要である。例えば、あるクラスタ内のあるサーバの CPU において熱暴走が発生した場合、そのサーバの電源を落とし、待機している他のサーバの電源を入れる、といった機能を提供する管理ソフトウェアの動作検証を行える必要がある。

さらに、複数台サーバ環境下には、同一のハードウェア構成ではなく、各々異なるハードウェア構成を持つサーバが存在するかも知れない。このような環境下での管理ソフトウェアの動作検証を行うことは、ディペンダブルシステムを実現する上で重要である。

4.2 設計概要

本故障シミュレータは、オープンソースのプロジェクトである OpenIPMI[10] で提供されている IPMI ドライバを利用しており、この IPMI ドライバの一部を変更して作成したハードウェア故障模擬モジュール、および、障害発生指示を与える障害発生指示ユーザインタフェースからなる。構成は、図 5 のようになる。

ハードウェア障害模擬モジュールは、ソフトウェア的に BMC の挙動をエミュレーションするものであり、IPMI で定められたプロトコルに従って、管理ソフトウェアに BMC が保持している情報を返す。障害発生指示ユーザインタフェースは、ハードウェア故障模擬モジュールにおいてエミュレーションされた BMC の内部パラメタを変更するユーザインタフェースを提供する。

4.3 故障シミュレータへのインタフェース

ここでは、管理ソフトウェアから見た故障シミュレー

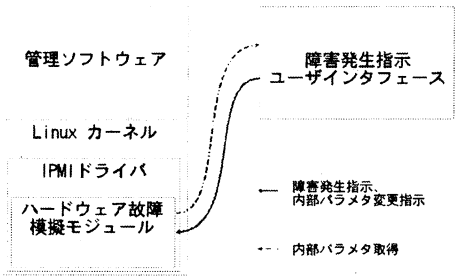

図 5　故障シミュレータの構成

タへのインタフェースについて述べる。前小節で述べた設計概要の枠組にて故障シミュレータの基本的な機能を管理ソフトウェアに提供できることを説明する。

前述のように、IPMI 規格において管理ソフトウェアが BMC へのアクセスを行うためのインタフェースとして、コンソールインタフェース、および、LAN インタフェースの 2 つが定められている。コンソールインタフェースにおいては、第 2.1 節で説明したように IPMI ドライバを介して BMC へのアクセスが行われるため、ハードウェア故障模擬モジュールにおいて、ソフトウェア的にエミュレーションされた BMC の情報を得ることができる。

LAN インタフェースを使用する場合、管理ソフトウェアは IPMI ドライバを介さず、直接 BMC にアクセスする仕様となっている。本故障シミュレータにおいては、LAN インタフェースを用いたアクセスを IPMI ドライバを介して行わせるようにするため LAN Emurator を使用する。LAN Emurator は、ソフトウェア的に BMC の LAN インタフェースを模擬するデーモンプロセスである。このプロセスを立ち上げることで、管理ソフトウェアは LAN インタフェースを用いる際、LAN Emurator、そして IPMI ドライバを経由して BMC にアクセスするようになる。したがって、コンソールインタフェース、および LAN インタフェースのどちらを使用する場合においても、管理ソフトウェア側には何ら変更を加えることなく、あたかも IPMI 規格に準拠しているハードウェア (BMC) にアクセスするように、本故障シミュレータにアクセスすることができる。

4.4　検証支援技術の設計

前小節で述べた機能は IPMI 規格に基づいたハードウェア (BMC) の動作をソフトウェア的にエミュレーションするために必要となる基本機能である。ここでは、本故障シミュレータにおける第 4.1 節で述べた機能の詳細について述べる。この機能により、実際の現場で本故障シミュレータを効率的、かつ効果的に活用することができる。

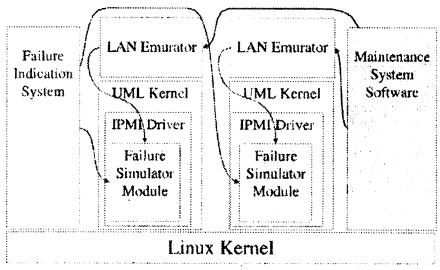

図 6　故障シミュレータ動作イメージ

4.4.1　ハードウェア状態のロギング、再現機能

本故障シミュレータにおいて、ハードウェア状態が変化するのは、障害発生指示ユーザインタフェースより BMC の内部パラメタの変更を行った時のみである。したがって、ハードウェア状態変化のロギングを行うには、パラメタ変更指示および、その時に発生したイベント情報として SEL、これらの情報をさらに細かく解析するために SDR および FRU (Field Replaceable Unit) 情報をファイルにロギングする。ハードウェア状態を再現する機能の実装は、BMC の内部パラメタの変更を行う機能を持った簡易なスクリプト言語を作成することによって実現する。障害発生指示ユーザインタフェースから行える操作に加え、故障発生のタイミングを記述できる機能を持つ。

4.4.2　複数台サーバのシミュレーション機能

本故障シミュレータでは複数のサーバ環境をシミュレーションするために UML (User Mode Linux)[5] を用いる。UML はユーザ空間で動作する Linux カーネルプロセスである。この UML 核にハードウェア故障模擬モジュールを組み込む。複数の UML プロセスを立ち上げることで、図 6 に示すよう、複数のサーバ環境をシミュレーションすることができる。

4.4.3　ハードウェア構成のエミュレーション機能

指定したハードウェア構成のエミュレーションを行うために、動作検証の対象となるハードウェア構成を持つホストもしくはファイルから情報を取得し、BMC 内部パラメタの変更を行うことで実現する。この機能を実現するために変更される BMC の内部パラメタは、ハードウェア構成をおさめている SDR 情報および各センサの現在の状態である。

本故障シミュレータは前述のようにソフトウェア的に BMC のエミュレーションを行うため、IPMI 規格に準拠していないハードウェア上においてもハードウェア障害のシミュレーションを行うことが可能である。さらに、この機能により様々なハードウェア構成を持つホスト上における障害発生をシミュレーションすることが容易となる。前述の UML を使用した複数台サーバ環境のシミュレーションと、この機能により、環境構築のための作業の工数およびその経費の大

幅な削減が期待できる。

5. 関連技術

5.1 機器健全度

個々の機器が正常稼働しているかどうかの指標をモデル化しているシステムがある。IBM 社の Tivoli ツール [14] では、ある期間、機器モニタが発生する Indicaion(Occurence) の数を用いて、イベントの発生ならびに機器の健全度 (Health) を定義している。イベントの発生は、Occurence の数と Hole (Indication なし) の数で定義される。例えば、3 つの Occurence と 2 つの Hole が生じた時にイベントが発生すると仮定すると

1 0 0 1 0 0 1

で、一つのイベントが発生することになる (ここで 1 は Occurence、0 は Hole)。以下の場合は、イベントが発生しない。

1 0 0 1 0 0 0

さらに、Occurence と Hole の数を使って、健全度 (Health) を定義している。100 はイベントが生じていないことであり完全に健全であると定義する。0 は、イベントが発生したことを意味する。健全度は、イベントが発生するために必要な Occurence に対して、現在までに生じた Occurence の割合で表現される。例えば、一つ目の例では、健全度は、66.6%となる。

Tenjin では、CIM のインディケーションを受けてプログラムによってイベントを発生させる機能を持っている。

5.2 CIMOM

DMTF で規格化しているデータベーススキーマは CIM (Common Information Model) と呼ばれ、CIM データベースサーバは CIMOM (CIM Object Manager) と呼ばれる [11]。CIMOM はプロバイダと呼ばれるインターフェイスを持ち、機器情報や OS からの情報をデータベースに登録する枠組を提供している。プロバイダには、SNMP や IPMI が想定されている。CIMOM の実装としては、SNIA CIMOM, openPegasus, openWBEM などがオープンソースとして入手可能である。これらは、問い合わせ型のインターフェイスしか提供していない。

5.2.1 WBEM

WBEM (Web Based Enterprise Management) では、CIMOM とのインターフェイスに http プロトコルを用い、オブジェクト授受のために xmlCIM と呼ばれるデータエンコーディングが規定されている。我々が知る限り、WBEM で実現されているのは、CIMOM 経由での CIM オブジェクトの閲覧機能、機器からのインディケーションを受け取る機能である。

6. おわりに

本稿では、現在開発中の自律的システム管理ソフトウェアならびに故障シミュレータについて述べた。本システムの管理保守ソフトウェアは、CIM における機器管理モデルに基づき、時制ルールとアクションにより自律的運用を支援する。故障シミュレータを使用することにより、運用者は運用ポリシが設計通りに稼働するかどうかを確認することが可能となる。

謝　辞

本研究の一部は、文部科学省「eSociety 基盤ソフトウェアの総合開発」の委託を受けた東京大学石川研究室および東京大学石川研究室と NEC ソフト株式会社との共同研究契約に基づいて行なわれた。

参考文献

1) Intel, Hewlett-Packard, NEC, and Dell, "- IPMI - Intelligent Platform Management Interface Specification, V1.5," 2002.
2) 石川、住元、久門、木村、岡家、「次世代高性能計算機アーキテクチャ上のシステムソフトウェア開発環境」、情報処理学会研究報告 03-OS-12 (SWOPP03)、情報処理学会、2003.
3) 住元、久門、石川、「10Gb Ethernet 上の通信プロトコル作成支援技術」、情報処理学会研究報告 03-OS-12 (SWOPP03)、情報処理学会、2003.
4) 岡家、木村、石川、「IPMI 規格に基づく管理保守系システムソフトウェア」情報処理学会研究報告 03-OS-12 (SWOPP03)、情報処理学会、2003.
5) http://user-mode-linux.sourceforge.net/
6) Hewlett-Packard、「Insight Manager 7 User Guide」、2002.
7) NEC、「White Paper NEC Express 5800 サーバ リモートマネジメント機能」2000.
8) NEC、「MWA Ver.3.10.xx ファーストステップガイド」第 46 版、2002.
9) Jean Arlat, Jean-Charles Fabre, Manuel Rodriguez and Frederic Salles, "Dependability of COTS Microkerlnel-Based Systems", IEEE Transactions on Computers, Vol. 51, No. 2, 2002
10) OpenIPMI, http://openipmi.sourceforge.net
11) DMTF, http://www.dmtf.org
12) Clusterworkx, http://www.linuxnetworx.com
13) Ganglia, http://ganglia.sourceforge.net
14) Stephen Hochstetler, Ghufran Shah, Jamie Carl, Jason Shamroski, John Willis, and Murilo Aguiar, "IBM Tivoli Monitoring Version 5.1: Advanced Resource Monitoring," IBM Redbooks, ISBN 0738426938.
15) DMTF, "CIM Concepts White Paper CIM Versions 2.4+," Document Version 0.9, DSP110, 2003.
16) Beehive, "The Book of Zope: How to Build and Deliver Web Applications," Orelly & Associates Inc, ISBN1886411573.
17) SNMP, http://www.ietf.org/rfc/rfc1157.txt