

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

秋山一人 鈴木康裕 津村直史 西村康孝

Autonomic Computing Architecture for Embedded Devices

Kazuhiro Akiyama, Yasuhiro Suzuki, Tadashi Tsumura and Yasutaka Nishimura

近年複雑化・高度化し IT リソースとして扱われる情報家電などの組み込みデバイスでは、オートノミック・コンピューティング (AC) の適用により効率的な管理ができることが期待されている。本論文ではデバイス固有の適用条件を考慮して、デバイスに適用可能な AC アーキテクチャーを提案する。さらに実際のユース・ケースに基づいたプロトタイピングを行い、考案したアーキテクチャーを検証した。実デバイスを用いた検証の結果、デバイスの自律管理、遠隔管理が効果的に行えること、デバイスの実行環境に合わせた柔軟性の高い管理が行えることが実証できた。

Embedded devices, such as information home appliances, have become more complicated and sophisticated in the past few years, and these devices can be managed more efficiently by using Autonomic Computing (AC). This paper proposes a new AC architecture that considers its applications for the device. The authors further verified this architecture by creating a prototype system using a real device. As a result of the verification, as expected we have shown that this architecture can achieve efficient self-management, remote management and flexible management.

Words & Phrases : オートノミック・コンピューティング, デバイス管理, リソース管理, 組み込みシステム, シンブトン
Autonomic Computing, Device Management, Resource Management, Embedded System, Symptom

1. はじめに

オートノミック・コンピューティング (以下 AC) とは、システムが自らを自律的に管理することで大規模化・複雑化したシステムを簡素化するアーキテクチャーであり、IT リソースを仮想化・抽象化して容易に管理することを可能にしている [1,2]。AC により人が関与する部分を減らして IT プロセスを自動化、高い可用性や柔軟性をもたらすことが可能になる。

近年、組み込み技術の高度化に伴い、携帯端末や情報家電をはじめ、プリンター複合機や自動販売機といった複雑化したデバイスが、IT リソースとして扱われるようになり、AC により効率的な遠隔管理が行えるようになるだけでなく、予防保守や自律管理が可能となり保守コストの削減に大きく貢献することが期待されている。また機能追加や拡張が日々行われるデバイスに対し、振る舞いを柔軟に変更できる AC により、新たな付加価値を生み出すことが可能になると考えられる。

一方で、現在考えられている AC は大規模 IT システムを想定しているため、潤沢なハードウェア・リソース、常時ネットワーク接続、特定の Java 環境など多数の適用条件が存在している [3,4,5]。

現状のデバイスにおいては使用できる機能・リソースが限定されているだけでなく、その多様な実行環境、想定されるユース・ケースにおいてこれらの前提条件を満たすことは困難である。さらにデバイスには AC では考慮されていない固有の適用条件が存在し、現在の AC アーキテクチャーをそのまま適用することは困難である。デバイスに適用可能とするためには、より柔軟で適用範囲の広い組み込み AC アーキテクチャーが必要である。

この論文では現状の AC アーキテクチャーをデバイスに適用する上で、デバイス固有の適用条件やユース・ケースに従った新たな組み込み AC アーキテクチャーを提案する。以下、2 章で現状の AC アーキテクチャーの概要と課題を紹介し、3 章で、その課題を解決するアーキテクチャーを検討する。さらに、4 章でプロトタイピング

提出日:2007年8月31日 再提出日:2008年6月30日

によるアーキテクチャーの有効性を検証・考察し、最後に5章で結論について述べる。

2. 現状の AC アーキテクチャーの課題

2.1 現状の AC アーキテクチャーの概要

現状の AC アーキテクチャーでは、管理対象リソースは、業界標準に基づくセンサーとエフェクターを備え、オートノミック・マネジャー (Autonomic Manager: AM) によって管理されると定義されている [1]。ここで、AM は、図 1 のように MAPE ループ (Monitoring, Analyzing, Planning, Execution) を構成する一連のモジュールを、ナレッジに基づいて制御し、リソースで発生したイベントを解析して症状 (シンプトン [10]: ナレッジの 1 つ) を検出し、対応する変更を計画し、アクションを実行することで、リソースの自律管理機能を実現する。これらの技術については、IBM 製品に取り込まれるとともに、IBM Support Assistant [6] の一部の機能として提供されている。また、センサー／エフェクターを実現する標準管理インターフェースとしては、Web-Services Distributed Management (WSDM) [7] が OASIS に

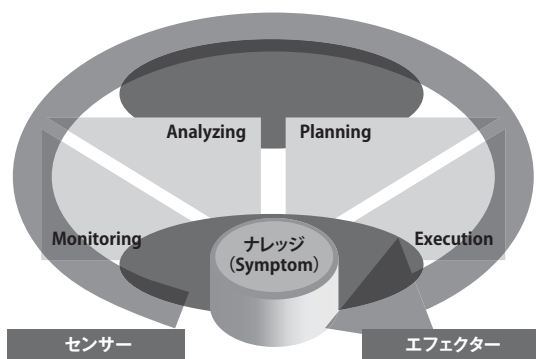


図 1. MAPE ループ構成図

より標準化されている。

MAPE ループの Monitoring 部分を実現するコンポーネントである Active Correlation Technology (ACT) [8] は複数イベントからなるシーケンスを元にイベント間の相関関係 (イベント・パターン) に基づく結果 (アクション) を出力するエンジンであり、上記 AM によりコントロールされる。ACT ではイベントの定義をプラグ・イン機構にすることによりさまざまなフォーマットに対応しており、Java 言語を利用した複雑なルール (ACT ルール) の記述を可能としている [9]。図 2 のように ACT はイベント・パターンを解析するエンジン、相関ルールを GUI で記述できるルール・エディター、ルールをエンジンに組み込む ACT コンパイラから構成されている。

2.2 デバイスへの適用における課題

現状の AC アーキテクチャーをデバイスに適用する上での課題について以下に示す。

① リソースの制約と前提条件

一般的にデバイスでは補助記憶容量に制約があるので、すべてのイベントやナレッジを保持することができない。同様に解析できるシンプトンの複雑さにも限界があり、多大なリソースを必要とする解析はデバイス上では行えない。また、ACT エンジンのアーキテクチャーでは、メモリー上にイベントとナレッジを保持しているため、リソースに制約のあるデバイスには実装が困難である。ACT ではナレッジを Java 言語で表記し、解析前にコンパイルする必要があるため、コンパイルと解析は機能的に同一な Java 環境で行う必要がある。デバイスとサーバーの Java 環境が異なる場合やデバイスが Java をサポートしない場合、サーバー上でナレッジを作成し、デバイス上で解析を行うという前提条件は満たせない。

② デバイスの多様性

一般的に、デバイスは多様な構成や環境で利用されるため、同じシンプトンであってもイベントの発生条件や対処方法がデバイス、またはそのときの実行環境によって異なる。現状の AC アーキテクチャーではこのデバイスの多様性に対応することができず、MAPE ループの動作自体を柔軟に動的に変更できるアーキテクチャーが必要である。

③ ネットワーク要件

現在の WSDM の実装は、リソース側に SOAP サーバーの実装が必要であり、サーバーとリソース

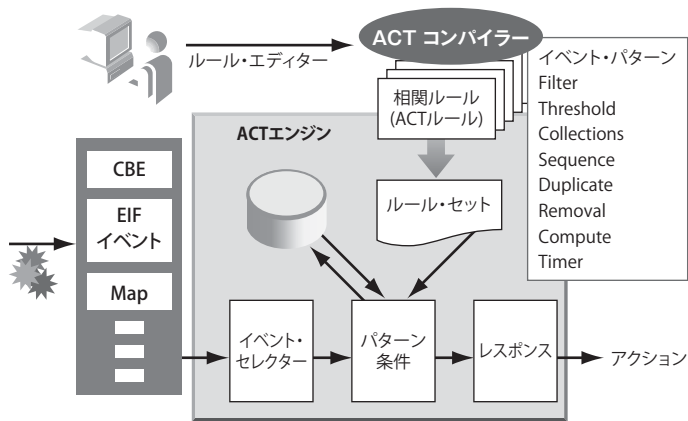


図 2. ACT 構成図

の間は常時接続されていて、必要なときはいつでもサーバーからリソースを操作できるシナリオ（ハンズ・オン）を想定している。このような要件を満たすデバイスは実環境ではほとんどない。

2.3 AC アーキテクチャーの拡張

前節の課題を解決するために、AC アーキテクチャーを次の2点で拡張する必要がある。

- さまざまなナレッジを持つ複数の AM 間の動作定義。
- 検出したシンプトンに対応するアクションの実行方法。

3. 組み込み AC アーキテクチャーの検討

3.1 Repository based Correlation Technology (RCT)

リソースに制約条件のあるデバイスに対して、メモリーを消費しないレポジトリ・ベースの解析エンジンを提案する。図3に構成図を示す。

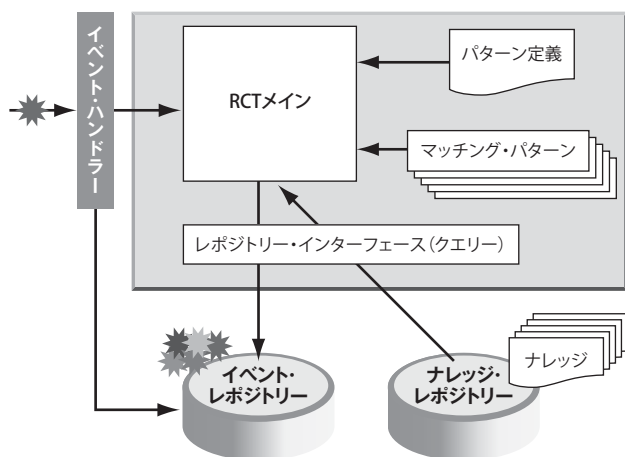


図3. RCTの構成図

RCTは、MAPEループにおけるMonitoringのモジュールであるACTと同等の機能を持つ組み込みデバイスに適した解析エンジンであり、従来のACTではメモリー上に展開していたナレッジやイベントをレポジトリ上に格納しておき、このレポジトリに対してクエリーを発行することで解析を行う。ナレッジに記述される相関ルールはSQLに基づく条件指定によるメタ・データ表記で記述し、条件はパターン定義(図3参照)によりサポートするマッチング・パターンを決定するプラグ・イン機構とする。イベント・ハンドラーはイベントに対してユーザーが定義可能なプリプロセスを実行し、イベントをレポジトリに格納する。イベントIDをRCTに渡し、RCT

は解析結果として検出したシンプトンを返す。

RCTの採用により以下の効果が期待される。

- 消費メモリーの削減
メモリー上にナレッジ、イベントを展開しないので消費メモリーを大幅に削減できる。
- 適応できる実行環境、シナリオの拡大
メモリー上で解析を行う場合、エンジンの停止・再起動により解析が停止してしまうという大きな問題があった。RCTでは静的なレポジトリに対して解析を行うため、エンジンが頻繁に停止・再起動される環境でも解析は継続して行える。またレポジトリが複数の解析エンジンでシェアできることから分散環境での使用も可能である。相関ルールをメタ・データ表記で記述するため、ACTで問題であったJava実行環境への依存は解消される。
- パターン定義のプラグ・イン化により、新しいマッチング・パターンを容易に追加実装できる。
- イベント・ハンドラーでは、ユーザー定義のイベント・プリプロセスが可能のため独自フォーマットのファイルやログにも対応できる。

3.2 階層化 AM

デバイスでの自律管理機能を実現するため、デバイス上にAMを実装する。しかしデバイス上のAMでは保持できるナレッジ、イベントの量に制約があり、解析できるナレッジの複雑さも限定されている。大量のナレッジを保存できるサーバー上のAM(上位AM)とデバイス上のAM(下位AM)間でイベントの転送、解析の委譲を行えばこの制約の問題が解決できる。また、上

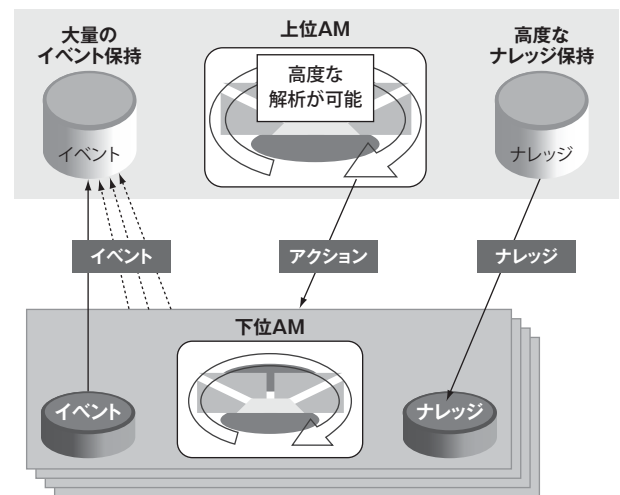


図4. 階層化されたAMの構成図

位 AM が複数のデバイスを管理している場合、これら複数のデバイスで発生したイベントを統合的に解析すれば、単独の AM では検出できないシンプトを検出することが可能になる。図 4 に階層化された AM の構成図を示す。

3.2.1 イベントの送信・解析の委譲

デバイス上で発生したイベントを上位 AM へ送信し、解析を委譲するインターフェースを定義した。イベントは以下のケースのとき上位 AM に送信され、解析が行われる。

- デバイス上でイベントが保持不可能
リソースの制約により保持できなくなったイベントを上位 AM に送信する。解析頻度の低いイベントや、上位で解析可能でデバイス上で即時応答性を求められない解析に使われるイベントから送信される。
- 上位 AM へ送信する必要があるイベント
イベントが重大で上位 AM に即時に伝える必要がある場合など、イベント・ポリシーにより上位 AM にイベントを送信する指定がされている場合に上位 AM に転送する。
- デバイスで解析不可能なイベント
複雑で高度な解析が必要でデバイスでは対応ができないイベント、ナレッジが存在せず上位でしか解析が行えないイベントなどが想定される。

3.2.2 アクションの送信

デバイスで発生したイベントが上位 AM で解析された結果、デバイス上で対応アクションを実行させる必要が生じる。このため WSDM を通じてアクションをデバイスの AM へ送信するインターフェースを定義した。

3.2.3 ナレッジの送信

上位 AM から更新されたナレッジを下位 AM に送信するインターフェースを定義した。ナレッジを下位 AM で作成・変更することは現実的ではなく、上位 AM で追加・変更され、他所から集約されたナレッジが上位 AM に蓄積され最適化されていく。これらのナレッジを下位 AM へ送信することにより、デバイスの自律管理がより効果的に行えるようになる。また、ナレッジを上位 AM で変更することにより、デバイスでのイベントに対するアクションや、下位 AM の動作などデバイスの振る舞いを容易に変更することが可能になる。

3.3 柔軟な MAPE ループの実装

多様な実行環境を持つデバイスに柔軟に対応するため、現在 IT 機器の管理で実用されているイベント・ハンドラー、解析エンジンに加え、図 5 で示すように新たにアクション決定エンジン、アクション実行時期決定エンジン、アクション・ハンドラーおよびこれらの間のインターフェースを MAPE ループ上に定義した。

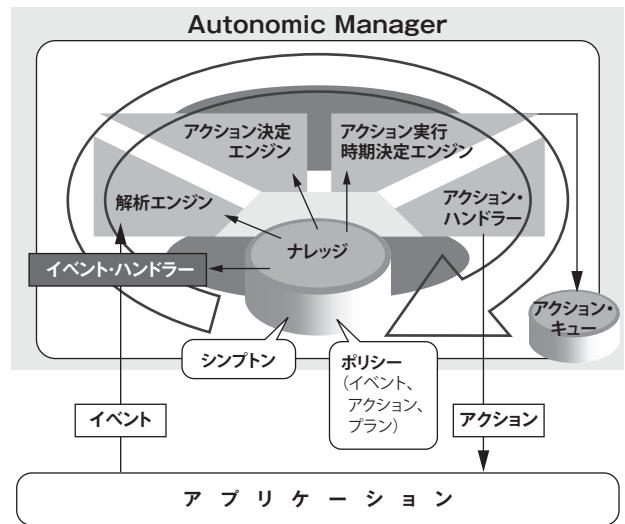


図 5. MAPE ループの実装

イベント・ハンドラー、アクション決定エンジン、アクション実行時期決定エンジンは、それぞれナレッジの 1 つであるポリシーによってその動作を指定できるよう定義する。それぞれのポリシーについて以下に述べる。

- イベント・ポリシー
イベント処理動作を動的に変更することを可能にした。イベントの属性およびデバイスの状況により、イベントに対して、無視、レポジトリーに格納、下位 AM で解析、上位 AM に送信などの処理指定が可能になる。
- アクション・ポリシー
デバイスではアクションは常に実行可能であるとは限らず、個々のデバイスによって実行できるアクション、できないアクションがあり、同じデバイス上でもそのときの状況により実行できない場合がある。検出されたアクションの実行可否をアクション決定エンジンにより動的に決定する。検出されたアクションに対して、実行する、実行しない、別のアクションを実行する、の指定が可能で、それぞれに判定の前提条件（デバイスの状態）が付加可能となる。

・プラン・ポリシー

デバイスではアクションが実行可能なタイミングが限られていることが多い。検出されたアクションをその実行環境でいつ実行するかをアクション実行時期決定エンジンにより動的に決定する。実行すべきアクションを直ちに行うか、遮断時、次の起動時、ネットワーク接続時、決まった日時などにスケジュールして行うかを指定可能とし、それぞれに前提条件が付加できる。スケジュールングを行うため、アクションを格納しておくアクション・キューを追加した。

3.4 ハンズ・オフのシナリオ

現実のデバイスが運用されているネットワーク要件では必ずしも上位 AM からデバイス側へ常に接続できるとは限らないため、デバイス側からアクセスするシナリオ（ハンズ・オフ）のサポートが必須である。現在考慮されていないこのシナリオを可能にするためのインターフェースを考案した。（図 6 参照）既存のハンズ・オンシナリオでは上位 AM（管理側）から下位 AM（デバイス）は常にアクセス可能になっている前提であり、アクションは即時に上位 AM から下位 AM に対して実行される。これに対し、新しく考案したアーキテクチャーでは、アクションは上位 AM 側のアクション・キューに格納され、下位 AM 側からのアクション受信要求によりアクション・キューから対象のデバイスに対するアクションが取得されて下位 AM に送信される。このアーキテクチャーではデバイス上に SOAP サーバーを実装する必要がなく、常時接続である必要もない。デバイスの都合の良いときにアクションを取得・実行することができる。

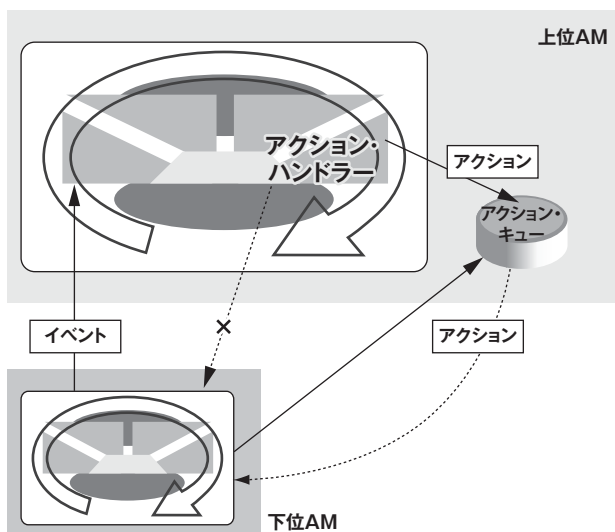


図 6. ハンズ・オフのアーキテクチャー

4. プロトタイピングによるアーキテクチャーの検証

4.1 プロトタイプの概要

図 7 にプロトタイプの構成図を示す。デバイスにはプリンター複合機を用いた。

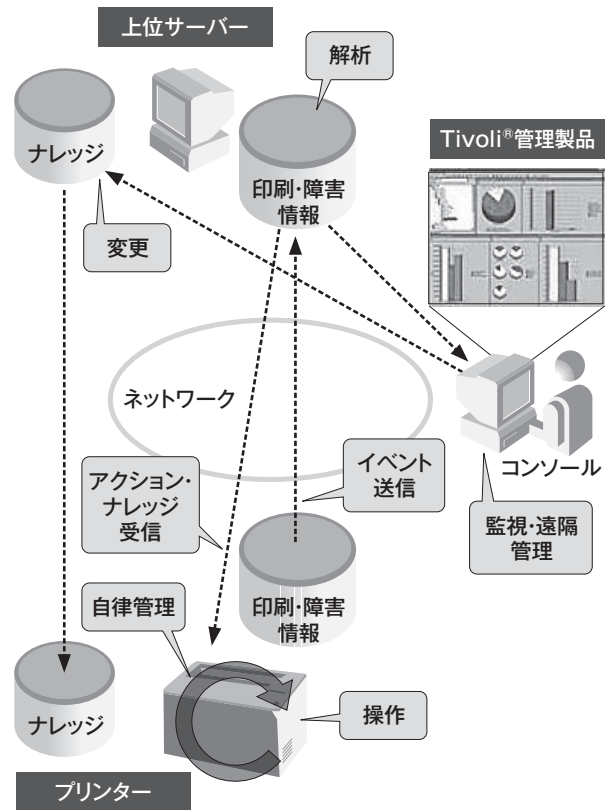


図 7. プロトタイプ構成図

上記の環境を使用し以下のユース・ケースで評価を行った。

- デバイス上で発生するイベントのうち、簡単に解析できるシンプトンについてデバイスで対処方法を判断し実行する。
- ユーザーの印刷操作をイベントとして発生させ下位 AM に格納し、定期的に上位 AM へ送信する。上位 AM によって解析が行われデバイスに対するアクションを実行する。
- 下位 AM では解析不可能な障害イベントを直ちに上位 AM に送信し、解析を委譲する。他 AM からの情報（イベント）を元に高度な解析を行い、代わりに使用可能なデバイスなどをデバイス上で表示させる。
- 自律管理（発生するイベントへの対応）に必要なナレッジを上位 AM で変更してデバイスの振る舞いを

- 変更する。
- 既存の IT 管理製品と連携し、オペレーターによるシームレスなデバイス管理を行う。

4.2 検証内容

4.2.1 RCT

RCT のリソース消費、保守の容易性、および適応性に関して検証を行った。

- RCT のフット・プリントは 100K 程度、メモリー消費は数百 K 程度で実行された。
- RCT におけるナレッジは SQL に基づく条件指定によってメタ・データとして記述できるため、ACT において必要とされていた Java 環境を必要とせず、上位 AM と同一のナレッジが使用できることを実証できた。図 8 に表記例の比較を示す。

拡張属性の“SUB_MSG”という文字列に Error という語を含むイベントを検出する例

◆ACTによるJava表記

```
act_event.getStringAttribute
("extendedDataElements.SUB_MSG")
.indexOf("Error")!=-1
```

◆RCTによるメタ・データ表記

```
<rule attribute="EVT_ATTRS.SUB_MSG"
value="%Error%"operation="like"
dataType="String"/>
```

図 8. ナレッジ表記の比較

- デバイスが停止・再起動された場合でも解析（自律管理）を継続することが可能であった。
- 簡単なクエリーで解析を実現しているため、低機能な組み込みデータベースである DB2® everyplace® [11] を使用したにも関わらず、デバイスの性能に影響を与えなかった。
- 従来サポートしていなかったマッチング・パターンに対しても容易に新しいパターンに対する実装を追加することができた。

4.2.2 階層化 AM

イベントの上位 AM への送信、解析の委譲、上位 AM で解析されたアクションの下位 AM への送信、および上位 AM で作成・変更されたナレッジのデバイスへの適用によって、以下のような自律管理、遠隔管理が実現可能であることを検証した。

- 下位 AM ではセルフ・クリーニングや軽度なエラーに対するメッセージ表示などの解析と対応を行い、以前の使用記録と比較した傾向分析や、障害発生時に他に使用可能な同等の機能を持つデバイスの検索といった高度な解析を上位 AM で行うといったシナリオが実現できた。
- 上位 AM で解析を行った結果、ユーザーの使用状況により、推奨される使用方法のデバイス・パネル上への表示、トナー濃度や省エネルギー・モードの設定をデバイスに行うことができた。
- 一定回数印刷をするとセルフ・クリーニングをする、トナー残量が一定以下になると警告メッセージを表示する、といった下位 AM のナレッジに対し、上位 AM で回数やしきい値を変更して、デバイスに送信することによりデバイスの振る舞いを変更できた。

4.2.3 MAPE ループ

デバイス上での自律管理が期待通り行えることを検証した。さらに上位 AM でポリシーを設定、送信することにより以下のような動作設定が可能であることを検証した。

[イベント・ポリシー]

- 通常の印刷操作はレポジトリーに格納するのみで解析は行わない。
- 重大度の高いエラーは上位 AM に送信する。
- ディスク空き容量が少ない時は、解析に使用しないイベントは無視する。
- CPU 使用率が高い場合にはイベントを上位 AM に送信し、解析を委譲する。

[アクション・ポリシー]

- 遠隔で設定を行わせないよう管理されているデバイスには、設定を行うアクションは実行せず、ログに記録するだけにする。
- ユーザーが印刷操作をしている場合にはアプリケーションの起動・停止を伴うアクションは行わない。

[プラン・ポリシー]

- アプリケーションを更新するアクションは次のデバイス再起動時に行う
- 蓄積された印刷情報イベントを上位 AM に送信するアクションは、深夜 2:00 ~ 3:00 のメンテナンス時間帯中でネットワークに接続されたときに実行する。

4.2.4 ハンズ・オフ

今回のプロトタイピングは、デバイスは常時接続ではなく、デバイス上には SOAP サーバーを実装しないというほぼ現実に近い環境で行った。この環境においてすべてのユース・ケースが問題なく実行でき、実環境への組み込み AC アーキテクチャーの適用ができることを検証できた。

4.2.5 IT 管理製品 (Tivoli) との統合

標準管理インターフェースである WSDM を用いたことにより、IT 管理で使用されている既存の IT 管理製品である IBM Tivoli 製品と容易に統合できることを検証できた。また WSDM によって、デバイスをネットワークに接続するだけで、管理製品に対して通知し、デバイスが自動的に管理対象リストに追加されるプロトタイプを行い、デバイスと IT 機器をシームレスに管理できることが検証できた。

4.3 考察

4 章の検証結果に基づき、考案したアーキテクチャーについての考察を以下に述べる。

- 考案した AC アーキテクチャーがプリンター複合機に対する効率的な自律管理が実証された。機能や実行環境、使われ方が多種多様で同じ種類であっても均一に管理が行えないデバイスにおいては自動化が困難で管理・保守に人が関与する部分が多い。このアーキテクチャーは、特定のデバイスに特化したものではなく、デバイスの多様性に柔軟に対応することが可能であり、また、階層化 AM やハンズ・オフといったユース・ケースを容易に実現することが可能である。
- RCT が軽量で保守の容易な解析エンジンであることが実証された。レポジトリの機能として分散・排他処理、複雑なクエリー、トランザクション管理などを使用することができるため、より複雑な解析やさまざまな環境での解析に対して柔軟な拡張性がある。
- デバイスの動作自体を可能な限りナレッジで制御する実装を行うことにより、デバイスの拡張がナレッジの変更のみで行えるようになった。そのため従来必要だったアプリケーションの置き換えや再起動が不要になり、AC の利点を生かすことができる。

5. おわりに

本論文では、組み込みデバイス固有の課題を解決した新たな AC アーキテクチャーを提案した。デバイスの課題に対し以下のような解決手法を示し、有効性を検証した。(1) リソースの制約・前提条件に対し、レポジトリを用いた解析エンジン (RCT) および階層化 AM を用いることで、複雑な相関ルールの対応、リソースの削減および実行環境の依存性を解消した。(2) デバイスの多様性に対し、MAPE ループにイベント・ポリシー、アクション・ポリシー、プラン・ポリシーを定義することで、さまざまな環境で動作するデバイスに対し動的に設定・変更することでより柔軟なデバイスの管理ができるようになった。(3) ネットワーク要件に対し、ハンズ・オフ・シナリオのサポートを追加し、デバイスが常時接続しない実際の環境をサポートすることができた。

考案したアーキテクチャーのプロトタイプによって、デバイスの自律管理、遠隔管理が効果的に行えること、デバイスの実行環境に合わせた柔軟性の高い管理が行えることが実証できた。このアーキテクチャーを適用することで、現在の IT リソース管理と用途や使用環境の点で大きく異なる産業機器の管理においても、IT リソースとして効果的に管理することが可能になると示された。

今回提案したアーキテクチャーの機能拡張によって以下のような分野に適用されることが期待される。(1) イベント・ナレッジの分散、解析の委譲、ナレッジの伝達の AM やナレッジが分散された環境への適用、(2) 考案されたアーキテクチャーが軽量で柔軟である、多様な製品の複雑な解析の要求に応えられることからデバイスに限らないベーシックな AM としての適用、(3) IT 管理製品との統合が容易であることから、IT 管理製品の新たなビジネス・エリアでの販売促進およびソリューションの獲得、(4) Java/SOAP の実装に限らないさまざまなデバイス (例えば C++) への AC アーキテクチャーの適用。

参考文献

- [1] IBM: "An Architectural Blueprint for Autonomic Computing," http://www.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf (2006).
- [2] S.R. White, et al.: "An architectural approach to autonomic computing," Proceedings of the 1st IEEE International Conference on Autonomic Computing, pp.2-9 (2004).
- [3] Y. Satoh, et al.: "Applying Autonomic Computing with Open Standardized Resource Interface WSDM to Managing Multi-vendor IT Systems," IM'07. 10th IFIP/IEEE International Symposium, pp.655-669 (2007).
- [4] H.M. Kreger: "Managing IT Resources Using Web Services: A Tutorial on the WSDM Standards From The Ground Up," NOMS 2006. 10th IEEE/IFIP International Symposium, p.585 (2006).
- [5] G. Dudley, et al.: "Automatic self-healing systems in a cross-product IT environment," Proceedings of the 1st IEEE International Conference on Autonomic Computing, pp.312-313 (2004).
- [6] IBM: IBM Support Assistant, <http://www.ibm.com/software/support/isa/>
- [7] OASIS Standard: Web Services Distributed Management, <http://www.oasis-open.org/specs/index.php#wsdmv1.1> (2006.8.1).
- [8] IBM developerworks: "Achieving complex event processing with Active Correlation Technology," <http://www.ibm.com/developerworks/autonomic/library/ac-act/> (2005.11.15).
- [9] IBM developerworks: "The autonomic computing edge: The role of knowledge in autonomic systems," <http://www.ibm.com/developerworks/autonomic/library/ac-edge6/index.html> (2005.9.13).
- [10] IBM developerworks: "The autonomic computing symptoms format," <http://www.ibm.com/developerworks/autonomic/library/ac-symptom1/> (2005.10.18).
- [11] IBM: DB2 Everyplace, <http://publib.boulder.ibm.com/infocenter/db2e/v9r1/index.jsp>



大和ソフトウェア開発研究所
ストラテジー
アドバイザリー・ソフトウェア開発エンジニア

秋山 一人 Kazuhito Akiyama

[プロフィール]

1990年、日本IBM入社。AS/400®のエミュレーター開発を経て、1995年よりシステム管理、デバイス管理を行うミドルウェア製品の開発に従事。2006年よりソフトウェア開発研究所にてオートノミック・コンピューティング関連製品の開発およびソリューション技術支援を担当。現在はストラテジー部門に所属。

akiy@jp.ibm.com



大和ソフトウェア開発研究所
クロスブランド・サービス&開発
シニア・テクニカル・スタッフ・メンバー

鈴木 康裕 Yasuhiro Suzuki

[プロフィール]

1985年日本IBM入社。ソフトウェアおよびハードウェアの製品保証、ソフトウェアの開発、技術サポート、サービス提供に従事。2004年より、オートノミック・コンピューティング技術、オンデマンド・コンピューティング技術を推進し、現在はクロスブランド・イニシアティブを担当。情報処理学会会員。

yazoo@jp.ibm.com



大和ソフトウェア開発研究所
Tivoli 先進技術推進
ソフトウェア開発エンジニア

津村 直史 Tadashi Tsumura

[プロフィール]

2003年日本アイ・ビー・エム入社。東京基礎研究所にて、オートノミック・コンピューティング、ITサービス・マネジメントの研究に従事。2006年からTivoli先進技術推進にて、オートノミック・コンピューティング関連のソリューション開発やTivoli製品の品質検証を担当し、現在はシステム管理ソリューションの技術支援を担当。

dashi@jp.ibm.com



大和ソフトウェア開発研究所
Tivoli 第一開発 ソフトウェア開発エンジニア

西村 康孝 Yasutaka Nishimura

[プロフィール]

2006年、日本IBM入社。以来、大和ソフトウェア開発研究所にてシステム管理製品のソリューション開発に従事。現在はTivoli第一開発にてサービス・マネジメント製品とアセット・マネジメント製品の品質検証を担当。

nishi2go@jp.ibm.com