ログによる動的なアシュアランス・ケース機構

岡本　悠希†　　内田　篤史†　　倉光　君郎†

† 横浜国立大学

E-mail: †{okamoto-yuki-gm,uchida-atsushi-vt,kimio}@ynu.jp

あらまし　アシュアランス・ケースは，安全工学分野から生まれた技術文書であり，根拠を示すことで，ディペンダビリティの確信を得ることに用いられる．従来，アシュアランス・ケースは，運用前にシステムのディペンダビリティを議論することに用いられており，開発時に行われるテストの結果等を根拠として用いていた．我々の目的は，アシュアランス・ケースと運用中のシステムのログを統合することで，運用時システムのディペンダビリティの議論にまでアシュアランス・ケースの利用範囲を拡大することである．本研究では，システムのログをアシュアランス・ケースと統合するためのログフォーマット，ログストレージの構成法について提案し，実際に運用中のシステムにおいてそれらの手法を利用して得られた知見について報告する．

キーワード　アシュアランス・ケース，ログ，モニタリング

1. はじめに

近年の情報システムの中には，社会基盤となり人々の生活にまで深く関わっているものも存在し，その信頼性や保全制，可用性などを総合したディペンダビリティが求められている．安全工学分野からの利用が進められているアシュアランス・ケース [1] は，従来から主に開発時のシステムのディペンダビリティ要求について議論することに用いられて来た．

近年では，アシュアランス・ケースをシステムの開発時のみでは無く，運用中のシステムのディペンダビリティについての議論にも利用しようという試みが見られる．

開発時のシステムに関するアシュアランス・ケースを記述する際には，その根拠となるエビデンスには，開発時のシステムにおけるソフトウェアテストの結果など，事前に用意できるデータが用いられる．

しかし，運用中のシステムに関するアシュアランス・ケースを記述する際には，その根拠となるエビデンスもまた運用中のシステムから取得されるものでなければならないが，システムの状態は刻一刻と変化するため，実際にアシュアランス・ケースが満たされているのかを把握することが困難である．

本研究の目的は，前述した問題を解決し，システム運用に関するアシュアランス・ケースの利用価値を高めることである．

我々は，システム運用におけるモニタリングログ収集の基盤として Runtime Evidence Collector（REC）の設計・実装を行い，収集したモニタリングログをアシュアランス・ケースのエビデンスとして利用可能な形にした．

本論文は，以下のとおりに構成される．2. 節では，一般的なシステム運用の技術について概観する．3. 節では，アシュアランス・ケースについての説明を行う．4. 節では，REC の設計について概観する．5. 節では，REC の実装について論じる．6. 節では，構築した REC を用いて収集したモニタリングログをアシュアランス・ケースのエビデンスとして利用する実験を行い，構築した REC についての評価を述べる．8. 節で，本論文を総括する．

2. システム運用

情報システムは社会基盤にまで成長し，人々にとって不可欠なものとなっている．近年ではこうした情報システムにおける障害が話題となり，社会に多大な影響を与えている．

こういった背景から，情報システムを適切に運用する技術が重要である．本節では，情報システムの運用技術についてまとめた．

2.1 システムのモニタリングとログ

システムを適切に運用するための技術として，古くからモニタリングが利用されている．モニタリングとは，定期的にシステムにおける様々なデータを計測しログとして記録することで，計測値が正常値を逸脱した際に異常として検出する技術である．モニタリングを行うことで，システムの異常をいち早く検知し，対処を行うことが可能になる場合がある．

また，モニタリングによってシステムの異常が検知できたとしても，異常の要因を特定できないことには対処が不可能な場合も存在する．そういった場合は，モニタリングによって記録されたログに対して解析を行うことで，異常の原因の特定を行う必要がある．

2.2 統合監視ツール

近年の情報システムは，扱う情報量の大規模化に伴うスケールアウトや，システムそのものの冗長化のため，複数のコンピュータによる構成の分散システムの形態を取っているものが大半を占めている．こういったシステムにおいてモニタリングを行う場合，それぞれのコンピュータでモニタリングを行った結果を集約するための仕組みが必要となる．

統合監視ツールでは，分散システムにおけるそれぞれのコンピュータで収集したモニタリングログを集約して管理することが可能である．ここでは，オープンソースで提供され広く用いられている Zabbix [2] を例に，統合監視ツールの役割について説明する．

Zabbix では，分散システムにおけるモニタリングによって収集されたログを，各コンピュータ上で動作する Zabbix Agent と呼ばれるアプリケーションが回収し，Zabbix Server でそれらのログを集中管理することが可能である．Zabbix は上記の仕組みで収集したログを用いて，以下の４つの機能を実現している．

- ネットワークを介して複数のホストの監視を行う（情報収集機能）
- 集めたデータを集中管理し，履歴やグラフ表示を行う（情報表示機能）
- 監視データに対して閾値を設定し，監視対象の正常/異常の検知を行う（閾値設定機能）
- 正常/異常の変化があった際に，何らかの手段でシステムの管理者に通知する（通知機能）

システムの管理者は，上記の機能を組み合わせて用いることで，システムの異常を検知することが可能になっている．

しかし，実システムでは様々な要因が作用しあうことで障害が発生する．例えば，統合監視ツールによって監視しているある項目で起きた異常が，他の監視項目で起きている異常の要因となっている場合も存在する．こうした障害発生における異常の伝搬については Avizienis による先行研究で詳細に説明されている．[3]

一方で，統合監視ツールはシステムの管理者が決めた項目を監視し，異常があった際に管理者に知らせるに過ぎないため，実際に障害対処を行う際には監視している項目で起きた異常が，どのように伝搬し障害に繋がったかについては，2.1 で述べたように解析を行うなどして，明確にする必要がある．

3. アシュアランス・ケース

アシュアランス・ケースとは，システムのディペンダビリティ要求を議論するための技術文書のことであり，安全工学分野から利用が広がっている．[4] [5]

アシュアランス・ケースは，特にシステムの開発におけるディペンダビリティを議論することに用いられてきたが，近年ではシステム運用に用いようという動向も見られる．[6]

本節では，3.1 で従来のアシュアランス・ケースの用いられ方について，3.2 でシステム運用に用いられるアシュアランス・ケースについて述べる．

3.1 従来のアシュアランス・ケース

従来からアシュアランス・ケースは，システムの開発におけるディペンダビリティを議論することに用いられており，その記述には Tim Kelly らが提唱するビジュアル表記法である Goal Structuring Notation (GSN) が用いられてきた．[7] 図 1 に GSN によって記述されたアシュアランス・ケースの例を示す．

GSN では議論を複数の要素に分解する．まず議論における主張をトップゴールとし，議論を分割する方針であるストラテジーによってトップゴールを複数のサブゴールに分割する．この手法でゴールを分解していき，末端にくるサブゴールにはエビデンスと呼ばれるゴールを満たすための証跡をつけることで，議論におけるトップゴールが保証されていることを示す．各ゴール，ストラテジー及びエビデンスの要素には，その要素が成り立つための前提条件をコンテクストと呼ばれる要素によって示すことができる．

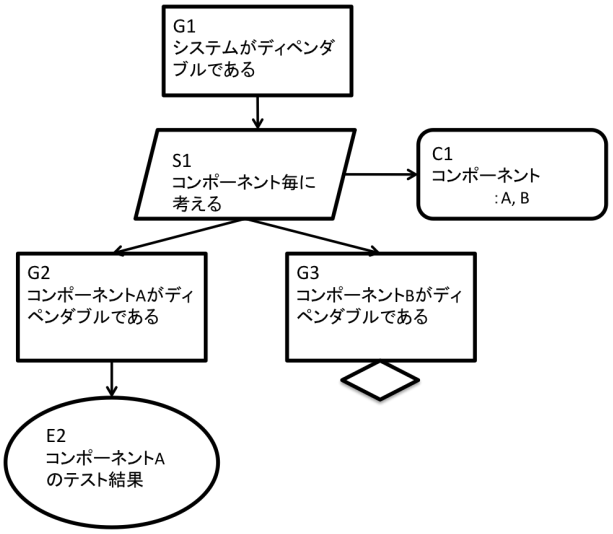


図 1　GSN で記述されたアシュアランス・ケース

また各要素は，種類によって表記する際の形を変えている．以下に各要素についての説明を示す．

- ゴール: 長方形で表し，システムについての議論の主張を示す
- ストラテジー: 平行四辺形で表し，ゴールを分割する際の方針を示す
- エビデンス: 楕円形で表し，ゴールが成り立つための根拠や証跡を示す
- コンテクスト: 角が丸くなった長方形で表し，各要素が成り立つための前提条件を示す
- アンデベロップド: 菱形で表し，エビデンスが存在しないことを示す

アシュアランス・ケースのユースケースについて，図 1 を例に考えてみると，まず開発するシステムについて，”システムがディペンダブルである”というトップゴールがあり，”コンポーネント毎に考える”というストラテジーのもと，複数のサブゴールに分割している．このとき，コンポーネントとは，コンポーネント A とコンポーネント B を表すことが前提条件としてコンテクストによって示されている．また分割された各サブゴールにおいて，サブゴール G2 の”コンポーネント A がディペンダブルである”という主張に対してはコンポーネント A のテスト結果という根拠が示されているが，サブゴール G3 の”コンポーネント B がディペンダブルである”主張に対しては，アンデベロップドが記されているため根拠が示されていない．

この例で示すように，システムの開発においてはシステムのサブゴールを満たすためのエビデンスとしては，システムのユニットテストの結果などのデータが用いられるのが一般的である．こうした文書の特徴としては，アシュアランス・ケースを記述する前に，事前に用意が可能であるという点が挙げられる．

3.2 システム運用に用いられるアシュアランス・ケース

3.1 で述べたように，従来からアシュアランス・ケースはシステムの開発におけるディペンダビリティを議論するのに用いられることが多かった．しかし最近では，システムの運用にも用いようという研究動向も見られるようになった．

前述したように，システムの開発に関するアシュアランス・ケースではサブゴールを満たすためのエビデンスとしてシステムのテスト結果など事前に収集できるデータを用いることができる．一方で，システム運用に関するアシュアランス・ケースのエビデンスには，システムを運用して初めて手に入るデータが利用される場合が多く見られる．システムの運用時に得られるデータというのは，基本的に 2.1 で述べたようなモニタリングで得られるログ等に相当する．システム運用に用いられるアシュアランス・ケースの例を図 2 示す．

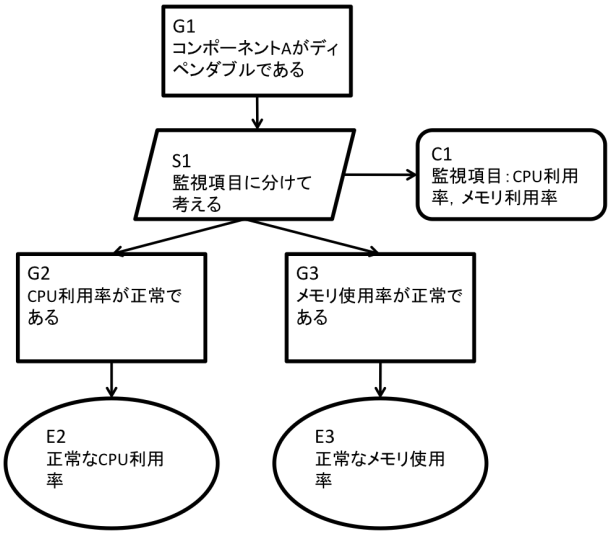


図 2　システム運用に用いられるアシュアランス・ケースの例

図 2 の例では，コンポーネント A のディペンダビリティについて監視項目に分けて議論している．ここで，監視項目に分割したサブゴールのエビデンスには正常な CPU 利用率や，正常なメモリ使用率などのデータが用いられているが，実際はこれらのデータはモニタリングによるログとして入手される．

システム運用におけるアシュアランス・ケースを記述することのメリットは，障害発生における異常の伝搬が一見して把握できる点である．アシュアランス・ケースを記述することで，統合監視ツールなどを用いてモニタリングを行っている一つ一つの監視項目が，システム全体とどのように関係しているかを明確にすることができる．図 2 の例だと，CPU 利用率の異常がコンポーネント A の異常に繋がることを一目見て理解することができる．

ここで問題となるのが，システムの状態が常に変化することである．一度システム運用に関するアシュアランス・ケースを記述したとしても，エビデンスとして用いるデータはシステムの運用時にモニタリングによって取得される最新のデータである必要があるため，常に更新されなければ記述されたアシュアランス・ケースが観測されたタイミングで満たされているのかがわからない．

また当然ではあるが，分散システム上において動作している多くのアプリケーションから出力されるモニタリングログの多くは，テキストファイルとして保存されるに過ぎないため，前述したような形でリアルタイムにアシュアランス・ケースにおけるエビデンスとして反映するには，必ずしも適したフォーマットであるとは言い難い．

4. Runtime Evidence Collector の設計

本研究の目的は，システムのモニタリングによって取得したログを，3.2 で述べたように，システムの運用について記述されたアシュアランス・ケースと統合することで，利用価値を高めることである．しかし，現状システム運用において収集しているモニタリングログは，そのままの状態でアシュアランス・ケースのエビデンスとして用いるにはギャップがあるため，適しているとは言い難い．モニタリングログをアシュアランス・ケースのエビデンスとして用いるのに適したフォーマットで記録及び利用するためのストレージ Runtime Evidence Collector (REC) を構築した．本節では Runtime Evidence Collector の設計について述べる．

4.1 設 計 要 求

我々は，現状のシステム運用において行われているモニタリングを踏まえた上で，REC の設計における要求として以下の 2 点を抽出した．

（1） 分散システムにおけるモニタリングログを扱うことが可能である

（2） リアルタイムにアシュアランス・ケースとの同期が可能である

まず一つ目の設計要求についてだが，2.2 でも述べたように，近年の情報システムは分散システムの形態を取っているものが大半であるため，統合監視ツールと同様に複数のコンピュータから集まってくるログを記録できる必要があると考えた．また現状で分散システムの監視のデファクトスタンダードともなっている統合監視ツールとも連携可能な設計にすることで，既存の分散システムにおいて取得されているモニタリングログも，アシュアランス・ケースのエビデンスとして利用できると考えた．

次に二つ目の設計要求についてだが，3.2 で述べたように，システム運用におけるアシュアランス・ケースでは，エビデンスと運用システムにおけるモニタリングで得られたデータが同期されている必要があると考えた．

4.2 以降では本節の設計要求に従った REC の設計についての説明を行う．

4.2 分散システムへの対応

設計要求で述べたように，REC は分散システムにおけるモニタリングに対応している必要がある．分散システムにおいては様々なコンピュータ言語で記述されたアプリケーションが様々な場所で動作しているため，そういった個々のアプリケーションから出力されるログも，統合的に扱えなければならない．

REC では，ログの記録及び取得の機能を Web API として

提供している．これらの機能を Web API として設計することのメリットとしては，モニタリングを行うアプリケーションにおいて，ログの記録のために特別にロギング用のライブラリを使用せずとも，コンピュータ言語側で HTTP 通信の機能さえサポートされていれば，ログの記録を行うことが可能である点である．図 3 に REC を用いた分散システムにおけるモニタリングの様子を示した．

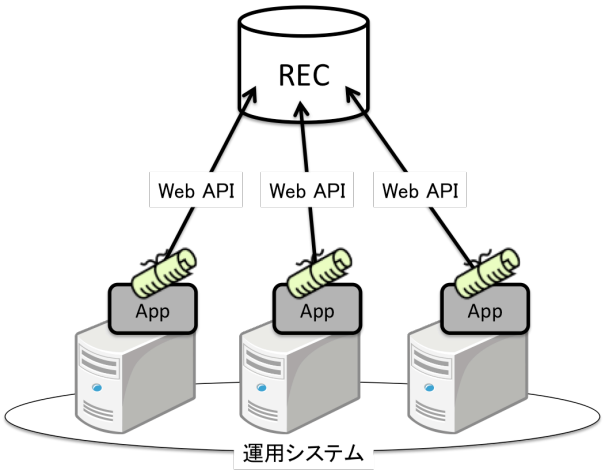


図 3　REC へのモニタリングログの記録

また，図 4 のような構成にすることで，統合監視ツールよって既に収集されているモニタリングログも REC で扱うことが可能な設計とした．

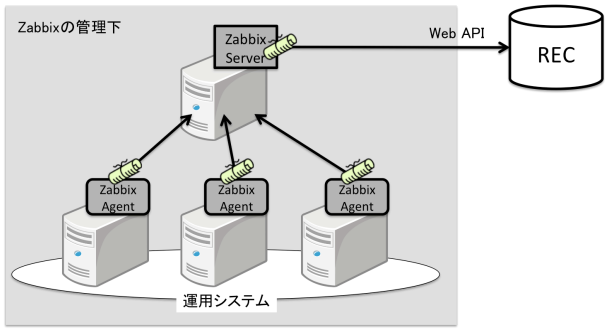


図 4　統合監視ツールと REC の連携

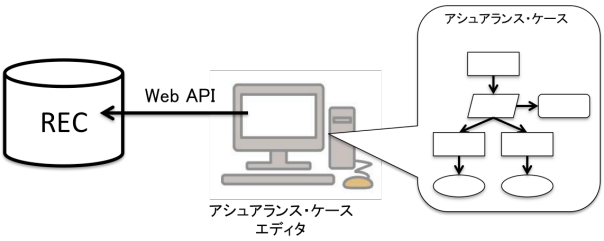
図 4 は，代表的な統合監視ツールの一つである Zabbix で収集したモニタリングログを REC から扱う際のシステム構成図である．Zabbix によって管理されている既存のシステム構成において，Zabbix Server が回収したモニタリングログを REC の Web API を用いて記録している．

4.3　アシュアランス・ケースとの同期

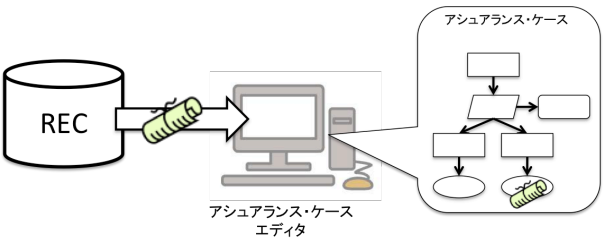
我々は，二つ目の設計要求に沿って，REC に収集したモニタリングログをリアルタイムにアシュアランス・ケースに反映させるための仕組みを設計した．

通常，アシュアランス・ケースの記述には専用のエディタが用いられる．[8] [9] 我々はこれらのエディタを用いて記述されたアシュアランス・ケースのエビデンスとして，REC に収集されたモニタリングログを用いることで，4.1 で述べた 2 つ目の設計要求を達成した．

4.2 で述べたように，REC に記録されたモニタリングログを利用する際には，Web API を使用して取得することが可能な設計となっている．このしくみをアシュアランス・ケース記述エディタから利用することによって，モニタリングログをアシュアランス・ケースのエビデンスとして利用することが可能である．図 5



(a) エティタアプリケーションからの Web API 呼び出し



(b) モニタリングログの取得・アシュアランス・ケースへの反映

図 5　モニタリングログとアシュアランス・ケースの同期

図 5(a) のように，定期的にエディタアプリケーションから Web API を呼び出し，取得したモニタリングログの値が正常かどうかを判定することで，図 5(b) のようにアシュアランス・ケースのエビデンスとしてモニタリングログを利用することができる．

5.　Runtime Evidence Collector の実装

本節では，4. 節で説明した設計に従った REC の実装について述べる．5.1 では，REC で扱うモニタリングログのフォーマットについて，5.2 では，ログの記録，取得に用いる API についての説明を行う．

5.1　ログフォーマット

REC にはアシュアランス・ケースのエビデンスとして用いるためのモニタリングログが記録される．故に，モニタリングログにはアシュアランス・ケースのエビデンスとして利用するために必要なデータが含まれていなければならない．我々はアシュアランス・ケースからモニタリングを利用する際に最低限必要なデータの項目として以下の 6 個を挙げた．

- type - 監視項目名（文字列型）
- location - モニタリングを行った場所（文字列型）
- authid - モニターの設置を行った人物を特定するための情報（文字列型）
- data - モニタリングによって取得された値（数値型）
- context - 任意の情報（JSON 型）
- timestamp - ログの記録時刻（時刻型）

type と location によってモニター（モニタリングを行うアプリケーション）が一意に特定される．例えば，サーバー A で CPU 利用率を監視するモニターが動いている場合，type が CPU 利用率，location がサーバー A となる．また authid は，そのモニタリングが誰の責任のもと行われたものかを示す役割の担う．これによって，アシュアランス・ケース上でモニタリングログをエビデンスとして用いる際に，そのログが正当なモニタリングによって取得されたものであることを示すことができる．具体的には，authid にはモニターを設置した人のメールアドレスを使用している．実際にモニタリングによって取得された値は data という項目に記録される．こちらの項目は数値型のみを許容しており，数値型で表せない情報に関しては JSON 形式で context と呼ばれる項目に記録される．

REC ではこれらの項目のデータを含むフォーマットでモニタリングログを管理している．

5.2 REC API

4.2 でも述べたが，REC へのログの記録及び取得は Web API を通して行うことが可能になっている．我々はこの Web API を REC API と呼んでいる．

REC API は JSON-RPC に準拠した形式での実装を行った. [10]JSON-RPC とは，JSON 形式でデータのやりとりを行うリモートプロシジャーコールの規格のことである．REC API では，5.1 で述べたフォーマットのモニタリングログを JSON 形式でやりとりすることによってログの記録及び取得を行う．図 6 に，REC API を用いて送受信するデータの例を示した．

```
{
        "jsonrpc": "2.0",
        "method": "pushRawData",
        "params": {
                "type": "CpuUsage",
                "location": "ServerA",
                "authid": xxx@gmail.com",
                "data": 45,
                "context": {}
        },
        "id": 0
}
```

(a) リクエスト

```
{
        "jsonrpc": "2.0",
        "result": { recid: 14 },
        "id": 0
}
```

(b) レスポンス

図 6　REC API を用いて送受信するデータの例

図 6 は，REC にログの記録を行う pushRawData という API を使用する際に送受信されるデータの例である．JSON-RPC では，データを送信する際に，method と呼ばれるフィールドに API の名称，params と呼ばれるフィールドに API の引数を与えてデータの送信を行う．図 6(a) の例では，params に 5.1 で述べた項目のデータを JSON 形式で与えることで，REC へのログの記録を行っている．params に timestamp は含まれていないが，timestamp についてはモニタリングログの記録時に REC で自動的に挿入される．API の戻り値は，受信するデータの result と呼ばれるフィールドに記録される．図 6(b) の例では，pushRawData の戻り値として recid と呼ばれる各モニタリングログを一意に識別するための ID を REC 側で自動生成して返している．

表 1 に pushRawData を含めた REC API の一覧を示す．

単純なモニタリングログの記録・取得以外にも，getRawDataList のように時間を指定して複数のモニタリングログを一度に取得する API や，getMonitorList のように現在 REC で管理しているモニター一覧を取得することができる API が用意されている．

6. 評　　価

我々は，実際に既存のアシュアランス・ケースエディタの機能を拡張し，REC と連携してモニタリングログをアシュアランス・ケースのエビデンスとして用いることが可能かどうかの評価を行った．

6.1 REC を利用したアシュアランス・ケースエディタの機能拡張

我々は，ウェブベースのアシュアランス・ケースエディタ Assure-It [11] に，モニタリングログをアシュアランス・ケースのエビデンスとして用いるために次の 2 つの機能を実装した．

- エビデンス確認機能
- エビデンス判定機能

まずエビデンス確認機能についてだが，こちらはアシュアランス・ケースのエビデンスとして用いられているモニタリングログを確認する機能である．この機能によりエディタのユーザーは，具体的にどのようなモニタリングログがエビデンスとして用いられているのかをエディタ上で確認することが可能である．

二つ目のエビデンス判定機能は，収集したモニタリングログが実際にゴールを保証するための根拠となっているかどうかを判定する機能である．この機能によりエディタのユーザは，記述したアシュアランス・ケースが現状で満たされているかをエディタ上で確認することが可能である．

6.2 評 価 実 験

6.1 で行った機能拡張済みの Assure-It を用いて，システム運用に関するディペンダビリティを議論するアシュアランス・ケースを記述し，モニタリングログをエビデンスとして利用する実験を行った．

6.2.1 実験の目的

本実験の目的は，運用中の実システムにおいて，REC を用いて機能拡張を行った Assure-It によって，実際に利用されている運用システムのモニタリングログをアシュアランス・ケースのエビデンスとして可能であることを確認することである．

6.2.2 実 験 内 容

本実験において，アシュアランス・ケースを記述しシステム運用に関するディペンダビリティを議論する対象としては，我々が運用するオンラインプログラミング演習システム ASPEN を

表 1　REC API 一覧

API 名	引数	戻り値	説明
pushRawData	type, location, authid, data, context	recid	モニタリングログの記録を行う
getRawData	recid	type, location, authid, data, context, timestamp, recid	モニタリングログの取得を行う
getLatestData	type, location	type, location, authid, data, context, timestamp, recid	最新のモニタリングログを取得する
getRawDataList	type, location, limit, start_timestamp, end_timestamp	[モニタリングログ]	ある時間帯におけるモニタリングログを配列として取得する
getMonitorList	無し	[モニター]	モニター一覧を配列として取得する

使用した．ASPEN は横浜国立大学や早稲田大学のプログラミングの授業で演習に用いられている実績がある．[12]

記述したアシュアランス・ケースは，大学における ASPEN を用いた 90 分の演習が正常に行われることに対するディペンダビリティを議論するものである．図 7 に実験に使用したアシュアランス・ケースを示した．

図 7 においては，演習が正常に行われるというゴールを，

- システムが正常に動作していること
- システムが提供するサービスによって生徒が演習で目的とした学習レベルに到達していること

という二つのサブゴールに大きく分けて議論している．

一つ目の，システムが正常に動作しているというサブゴールに対するエビデンスとしては，システムを構成するサーバー毎に死活監視を行い，そこで取得されたモニタリングログを使用している．Aspen のシステム構成図を図 8 に示す．

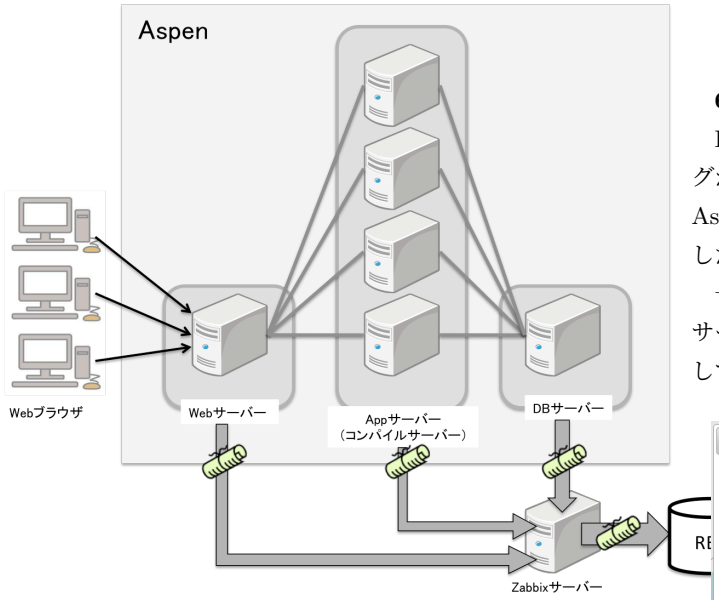


図 8　Aspen のシステム構成図

また二つ目の，システムが提供するサービスによって，生徒が演習で目的としている学習レベルに到達している，というサブゴールに対しては，生徒の Aspen 利用におけるアクティビティに関するログをエビデンスとして用いている．Aspen では，生徒がブラウザ上で記述したプログラムを図 8 における App サーバーでコンパイルし，その実行結果をブラウザに返して表示することで，ブラウザ上でプログラミングを経験することが可能になっている．Aspen の App サーバーにおいてプログラムのコンパイル時に発生するエラー情報から，生徒のプログラムの記述ミスをモニタリングすることで得られたログをエビデンスとして利用した．以下に今回のアシュアランス・ケースのエビデンスとして利用した監視項目の一覧を示す．

- システムが正常に動作していることに対するエビデンス
  - Web サーバー（1 台）の死活監視
  - DB サーバー（1 台）の死活監視
  - App サーバー（4 台）の死活監視
- システムが提供するサービスによって生徒が演習で目的とした学習レベルに到達していることに対するエビデンス
  - 生徒が記述したプログラムの文末に「;」を付け忘れていないかの監視
  - 生徒が記述したプログラムにおける変数宣言のし忘れの監視
  - 構造体の扱いにおけるコンパイルエラーの監視
  - 配列の初期化に関するコンパイルエラーの監視

6.2.3　実 験 結 果

REC には演習を行った 90 分間で 630 件のモニタリングログが記録され，これらのモニタリングログをリアルタイムに Assure-It で拡張した二つの機能から利用できることを確認した．

一つ目の拡張機能であるエビデンス確認機能を用いて App サーバーの死活監視によって得られたモニタリングログを確認している様子を図 9 に示した．

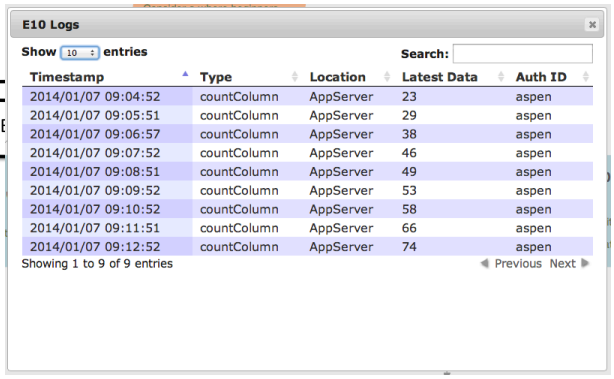


図 9　エビデンスとして用いられているモニタリングログ

また，二つ目の拡張機能であるエビデンス判定機能によって，

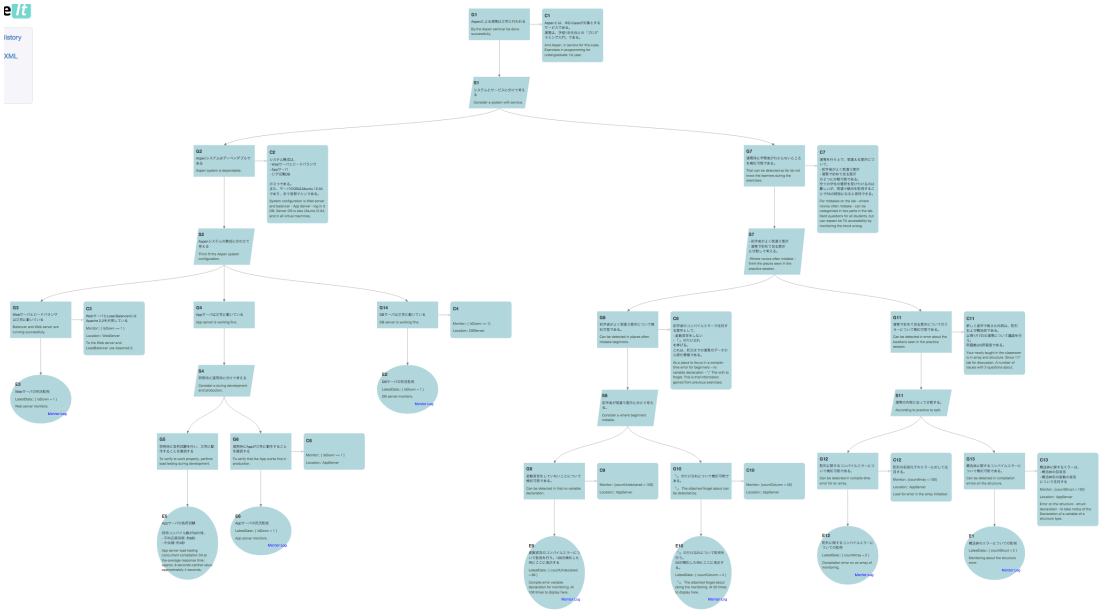


図 7　Aspen を用いた演習に関するアシュアランス・ケース

現在モニタリングログがゴールを保証するための根拠となっているかを判定している様子を図 10 に示した．

実際に取得したモニタリングログが満たすべきゴールの根拠となっていない際には，要素が赤くなり視覚的にエビデンスが妥当でないことを示している．

6.2.4 考　　察

本実験では，運用中の実システムである Aspen に関するアシュアランス・ケースにおいて，モニタリングログをエビデンスとして利用できることが確認できた．図 9，図 10 のようにリアルタイムにシステムの状態がアシュアランス・ケースから確認できるようになることで，3.2 で述べたような，現在用いられているシステム運用に関するアシュアランス・ケースの課題点を解決しているといえる．

しかし，今回アシュアランス・ケースを用いた議論の対象とした Aspen は，比較的小規模なシステムであり，記述したアシュアランス・ケース自体もそこまで規模の大きなものにはならなかった．今後，より大規模なシステムにおいても，REC を用いてアシュアランス・ケースからモニタリングログを利用しようと考えると，REC 自体がシステムのスケールアウトに対応していく必要があると考えられる．

7. 関 連 研 究

本研究と同様にシステム運用についてのアシュアランス・ケースを扱っている研究として，松野氏らによる A Framework for Dependability Consensus Building and In-Operation Assurance [6] がある．モニタリングログ等の実システムの運用時に得られた情報をアシュアランス・ケースにフィードバックし，エビデンスとして用いるというコンセプトにおいては我々の研究と類似している．しかし，彼らの研究では，エビデンスとして用いられるモニタリングログは，専用のツールを用いて収集したものでなければならないのに対し，我々の提案する手法では既存の統合監視ツールなどで収集されたモニタリングログを使用できるという点で異なっていると言える．

8. お わ り に

本研究では，従来開発時システムのディペンダビリティを議論することに用いられてきたアシュアランス・ケースの利用範囲を，運用時システムにまで拡大することを目的とし，システムのモニタリングログをアシュアランス・ケースのエビデンスとして利用するためのストレージ REC を提案した．また，我々で運用する実システムに関するアシュアランス・ケースを記述し，REC で収集したモニタリングログをエビデンスとして用いることで，ディペンダビリティについての議論を行う実証実験を行った．実証実験では，ASPEN 程度の規模のシステムにおいて REC を用いて収集したモニタリングログはアシュアランス・ケースのエビデンスとして十分に利用可能であることが示せた．

文　　献

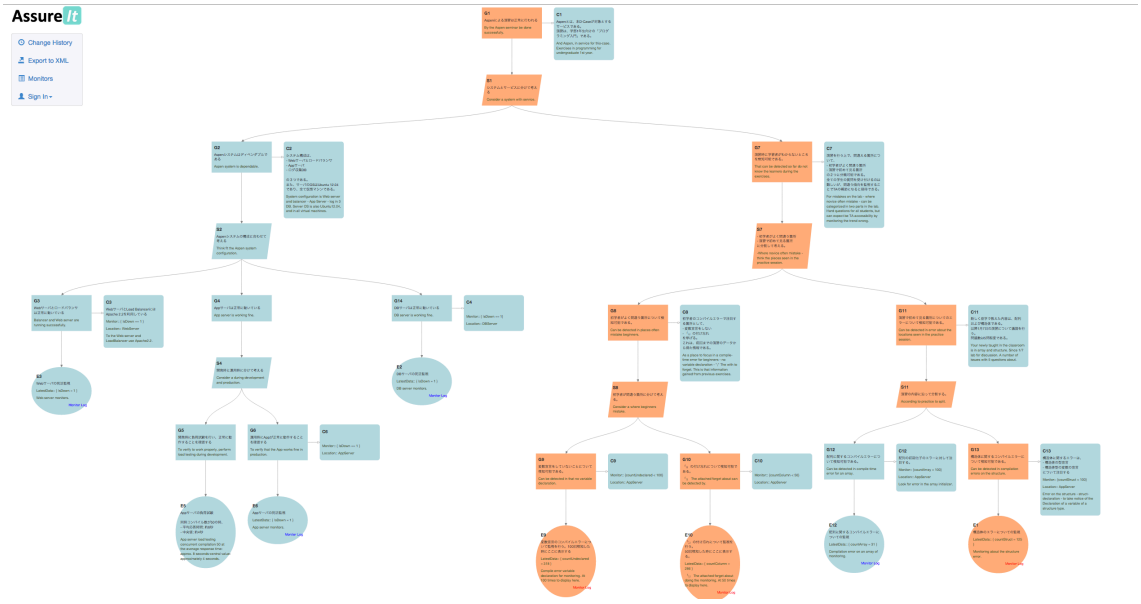
[1] JST CREST DEOS Project. *Safety Assurance Case* ガイド *(Ver. 1.0)*, 3 2012.

[2] Zabbix. `http://www.zabbix.com`.

[3] Algirdas Avizienis Ucla, Algirdas Avizienis, Jean claude Laprie, and Brian Randell. Fundamental concepts of dependability, 2001.

[4] Eunkyoung Jee, Insup Lee, and Oleg Sokolsky. Assurance cases in model-driven development of the pacemaker software. In *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 343–356. Springer, 2010.

[5] José Luis Vivas, Isaac Agudo, and Javier López. A method-



(a) 全体図

(b) 拡大図

図 10　エビデンス判定機能

ology for security assurance-driven system development. *Requirements Engineering*, 16(1):55–73, 2011.

[6] Yutaka Matsuno and Shuichiro Yamamoto. A framework for dependability consensus building and in-operation assurance. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 4(1):118–134, 3 2013.

[7] Tim Kelly and Rob Weaver. The goal structuring notation a safety argument notation. In *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[8] D-case editor. `http://www.dependable-os.net/tech/D-CaseEditor/`.

[9] Asce tools. `http://www.adelard.com/asce/index.html`.

[10] Json-rpc. `http://json-rpc.org`.

[11] Shunsuke Shida, Atsushi Uchida, and Kimio Kuramitsu. Assure-it: An administration tool for ensuring runtime synchronization of assurance cases. In *IEICE Tech. Rep.*, volume 113 of *DC2013-19*, pages 15–19, Fukuoka, Aug. 2013. Thu, Aug 1, 2013 - Fri, Aug 2 : Kitakyushu-Kokusai-Kaigijyo (DC, CPSY).

[12] Midori Sugaya, Takuma Wakamori, and Kimio Kuramitsu. Aspen: 自動データ収集機能を備えた web ベースプログラミング学習システム. In **情報処理学会シンポジウム論文集**, volume 2012, pages 3–8, Aug. 2012.