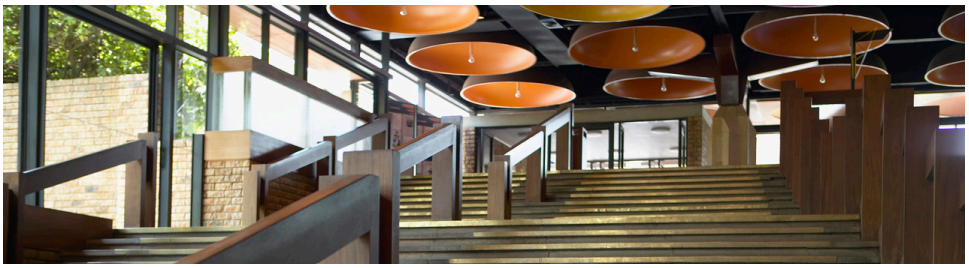
D-Script

スクリプトによる障害対応の実現

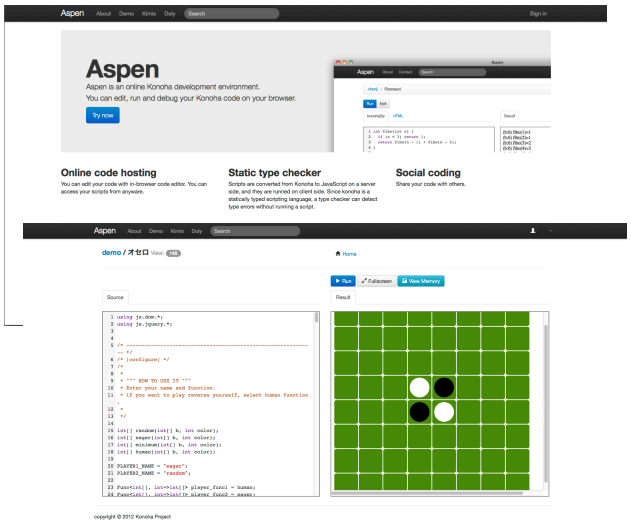
D-Script 研究チーム
横浜国立大学　倉光君郎



スクリプトによる柔軟かつ信頼できる障害対応の実現に向けて

　現代社会における情報化の進展は、コンピュータシステムの障害による社会への影響を深刻化させています。とくに、近年のシステム障害では、システム管理や障害対応のために用意されたスクリプト処理の不具合が増えています。たとえば、証券取引システムではバックアップサーバの起動スクリプトの応答がなく、複数のバックアップが起動してしまう事態[1]、クラウドサービスでは、個人情報保護のスクリプトが必要なデータまで消去する事態[2]があったと報告されています。

　このような事態の背景には、個々のソフトウェア開発は体系だって品質管理されているのに対し、最終的にサービスを統合しているスクリプトの作成は現場のエンジニアにまかされて、品質管理体制の体系化が遅れていることがあげられます。戦略的創造研究推進事業　CREST 研究領域「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」では、DEOS プロセスとスクリプト運用を統合させ、高信頼スクリプト技術 D-Script として、それらの運用を体系的にサポートする研究開発を進めています。本文では、こうした問題にいかに対応するか、わかりやすい実例をまじえてご紹介します。



ASPEN の障害事例に学ぶ障害対応とスクリプトの課題

ASPEN は、YNU 大学で開発された Web ベースのオンライン学習システムです。学生は、自宅のパソコンからサーバに接続して、プログラミングの宿題を解答できるように作られています。

今回のシナリオの主人公であるドリーは、ASPEN システムを積極的に売り込みました。売り込みの功が奏して、W 大学の「プログラミング演習」で利用してもらうことに成功しました。さあ、何事もトラブルなく演習はできるのでしょうか？

登場人物の紹介

ドリー	ASPEN のサービス提供の（一応）総責任者。自信にあふれ、細かいことは考えずに前に進む。
せんせい	ドリーからの猛烈な売り込みで ASPEN を授業で使うことに。
みょーあん	（いまひとつスキルと熱意が足りない）システム運用者。
ちぇんじぃ	（トラブルで呼び出された）ASPEN の開発者。思い込んだら一途なところがあり、空回り。
匿名希望	DEOS コンサルタント。 同窓会でたまたま会ったドリーに DEOS を紹介したことが、因縁の始まり。毒舌。

障害事例は実話にもとづいていますが、登場人物は実在の人物とは一切関係ありません。

サービス継続シナリオ未然回避

ドリーは、運用担当者のみょーあんと開発者のちぇんじぃを呼び出した。ASPEN で起こりうるトラブルを検討し、サービス継続シナリオを確認するためだ。

ドリー　W 大の演習に ASPEN を売り込んだよ。大丈夫だよね？

みょーあん　ドリーさん、いつも話が急ですね。学生さんは何人ですか？

ドリー　40 人くらいらしいよ。W 大の学生は、まじめだから出席率はいいと思うよ。サーバ負荷は大丈夫かしら？

ちぇんじぃ　（ASPEN のシステム構成図を示しながら、）ロードバランサーによる負荷分散を行い、サーバ台数が増やせるようになっています。何人にふくれあがろうが、理論的には対応できると思います。

ドリー　それは心強いね。もっとも、それより、学生の提出したプログラム課題が消えた方が痛いね。定期的なバックアップをお願いね。

みょーあん　わかりました。授業時間が終わるごとにバックアップをとるようにスクリプトを書いておきます。

大丈夫です。ASPEN はロードバランサーで負荷分散を行い、サーバ台数を増やして負荷対応できます。（ちぇんじぃ氏）

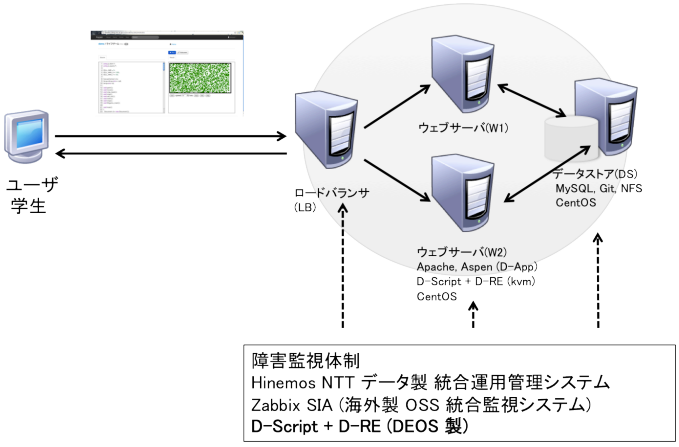


図 1　ASPEN のシステム構成図

（続く）

トラブル発生

今日は最初の授業だから、”hello, world”の表示だ。学生たちは、ちゃんと理解して表示できるかな？と思ったら…。

Proxy Error

The proxy server received an invalid response from an upstream server.
The proxy server could not handle the request *GET /account/blogs/*.

Reason: **Error reading from remote server**

Additionally, a 404 Not Found error was encountered while trying to use an ErrorDocum

Apache Server at www.technorati.com Port 80

図２ hello, world の代わりに表示された画面

せんせいは、授業中に ASPEN が正しく動作しないことに気づいて、運用者に連絡をとった。

せんせい	（電話で）ASPEN の運用者さんですか？ Proxy Error って表示されて、動作しないだけど…。
みょーあん	ミョーですね。そんなトラブル報告は、始めてですね。そちらのネットワークがおかしいのではないのでしょうか？
せんせい	そんなわけないでしょう。なんとかしてください。
みょーあん	（カタカタ）うーん。バーチャルマシンが落ちていますね。(手動で)再起動してみます。直りましたか？
せんせい	直ったみたいです。今日は、初日だから良いけど、次回からはこういうことが起きないようにしてください。
みょーあん	次回からは気をつけます。(なんで、バーチャルマシンは落ちたのだろう？？次回も落ちたら、ドリーさんに報告しよう。)

せんせいから、更なるキンキン声の苦情がきた。学生がプログラミング演習課題を実行しようとする度に、Proxy Error が表示されて演習が進まないそうだ。

障害原因を追求する

みょーあん　どうもミョーな事態ですが、W 大の授業中にサーバがダウンしているみたいです。

ドリー　何が起きているの？

みょーあん　…開発者の意見は？

ちぇんじぃ　ソフトウェアテストしたときは、ちゃんと動作していたので、バグではないと思います。サーバの異常負荷によるシステム障害かと思います…。

みょーあん　監視モニターのログを見る限り、異常な負荷は見当たらないなあ。システムは、何の事前の兆候もなく、突然、ダウンしています。そのとき、モニターも、一緒にダウンするので…あとはわかりません。

ドリー　せんせいの話では、講義の説明したあと、学生は一斉にプログラムを実行するらしいよ。

ちぇんじぃ　そうか、同時に 40 人アクセスがくるのか？それは想定していなかったですね。

ドリー　しかし、バーチャルマシン上の OS ごと死んでしまうのは、OS 専門家からみても不思議ねえ。障害の根本要因は何かしら。何か、ヒントはないの？

ちぇんじぃ　実は、D-Script でモニタースクリプトを追加して、ログを取ってあります。ちょっとお時間を頂ければ、結果を表示するようちぇんじぃしてみます。

ちぇんじぃ　お待たせしました。どうやらプロセスが大量に生成されているようです。どうして、大量のプロセス生成されているのかまではわかりません。

ドリー　うーん、結局、根本原因はわからないということね。

みょーあん　せんせいからのクレームもキツくなっているので…そろそろ。

ドリー　当初の予定どおり、サーバ増設を始めましょう。

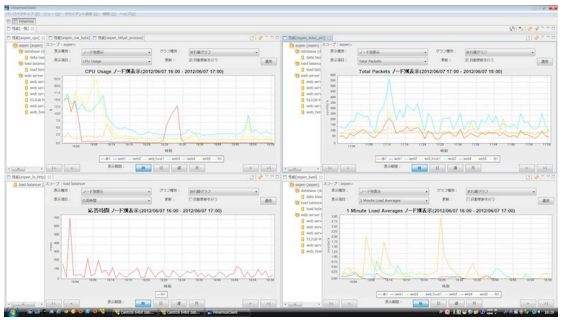


図 3　モニタリングシステム

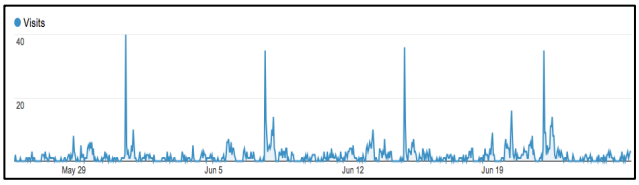


図 4　D-Script ログ解析の結果

想定どおり進まない対策

そのとき、みょーあん、ちぇんじぃはより深刻な問題を抱えていた。

ドリー　せんせいとの友情を失いそうなのだけど、さっさとサーバを増設しない？

みょーあん　ビミョーな話ですが、サーバ 1 台につき、同時アクセスで最大 4 人くらいしか処理できません。ということは、10 台程度のサーバ機を新調する必要があります。

ドリー　えーー、コスト増は勘弁して欲しいのだけど…。

ちぇんじぃ　もうひとつ問題があります。

ドリー　これ以上、聞きたくないわ。

ちぇんじぃ　ASPEN では、学生のプログラム保存用に GIT レポジトリを採用しています。そのため、I/O ボトルネックは残ってしまう可能性があります。

ドリー　つまり？

ちぇんじぃ　サーバを増設しても意味はありません。分散ファイルシステムの共有設定を非同期にちぇんじぃしてみましょう。それで解決するのではないかと思います。

みょーあん　非同期にしたら、保存されたデータが消えちゃわない？

ちぇんじぃ　（相手をにらみながら、）運がわるいときです。Google で調べた限りでは…。

みょーあん　それより、どうして最初から SQL サーバを使わないかったんだい？作りなした方がよくない？（にらみ返す）

せんせいは、イライラしていた。経験からいえば、あきらかに、サーバの処理能力不足だ。一体、いつサーバを増設する気なのだろう。

ドリーは、事態の深刻さに気づいた。いざっとなったら、サーバを増やせば、何人でも負荷に対応できると思っていたのに、どうやらそうではないらしい。

匿名希望は、破格のコンサルタント料を取ることで業界に名前が知れ渡っている。今回は、同窓会でドリーに DEOS を紹介したことがきっかけで、タダで相談に乗ることに…。

DEOS コンサルタント登場

ドリー　ちょっと、どうしてくれるのよ。どう責任とってくれるの？

匿名希望　（こういうクライアントは多いんだよね。）まずは D-Case をみせてよ。話はそれからだ。

ドリー　D-Case ？えー、D-Case？？？

匿名希望　やっぱり、D-Case を書いてないと思ったよ。（苦笑）まあ、今回、ラッキーなのは、開発者も運用者も身近にいるので、D-Case がなくてもある程度ヒアリングで状況はつかめるよ。もし何年も前に開発されたシステムだったら、D-Case がないと、まったくお手上げだったけどね。

ドリー　よかった。

匿名希望　で、安心するのはまだ早い。僕が整頓してみたところ…（表 1 を広げる。）

レベル	障害要因
要求レベル	そもそも在宅学習の利用を前提とし、演習室からの同時アクセスは想定していなかった（➔ ただし，スケールする設計を採用したはず）
設計レベル	CPU資源はスケールするが，I/Oはスケールしにくい設計だった
実装レベル	データの管理をMySQLではなく，GITで実装．(これが、性能低下の主因)
設定レベル	NFSのデータ共有の一貫性が同期モードであった．

表 1　DEOS コンサルタントによる障害原因のまとめ

ドリー　ほうほう。複合的な要因が絡み合った大事故だったというわけね。

匿名希望　そうじゃなくて、D-Case がないから、SQL サーバを使わない理由も、同期モードで運用している理由も、そこに至った議論も根拠もまったくわからないということだ。

ドリー　ということは？

匿名希望　仮に、ちぇんじぃ氏の主張どおり、運用を非同期モードに変更しようと考えても、その結果に対して何の保証もないよ。

ドリー　今からテストしてみるのは？

匿名希望　まさに「泥棒をみて縄を編む」というやつだね。

ドリー　意地悪いわないでよ。

匿名希望　D-Case がないから、障害対応の選択肢が狭まっていることは認識してよ。今からでも遅くないから、議論や根拠を D-Case で記述して、ステークホルダ間で障害対応についてちゃんと合意形成をしていくことが大事だよ。データを消しちゃったら、二度と顧客の信用は取り戻せないよ。

ドリー　なるほど。

D-Case がないと、誰も納得しませんよ。

暴走する現場

ファイルサーバの非同期モードで運用すれば、万事解決だ。ちぇんじぃは、設定を変更するスクリプトを新しく書いて、実行してみた。

みょーあん　ミョーだな？変な警告がでたぞ。意味不明だ。ドリーさんを呼ぼう。

ドリー　さっそく、ネズミが捕まったみたいね。

みょーあん　何をしかけたのですか？

ドリー　昨日、DEOS コンサルタントからアドバイスを受けて、「合意形成」が重要だという話だったから。

みょーあん　いよいよ買ってしまいましたか？
D-Case Editor？

ドリー　それはフリーだから買わないわよ。じゃなくて、D-ADD と合意形成支援システムを購入してしまったのよ。S 社の製品はこれが安くないのよ。

みょーあん　で、この結果ですか？

ドリー　そう。D-Script Engine のプラグインが付属して、D-Case 認証というアクセス制御を強化できるの。これから、ステークホルダ間で合意されていないスクリプトは、実行できないというわけよ。

ちぇんじぃ　そういうわけか…（ボソ）。

ドリー　ちぇんじぃ。（怖い顔して）
あなたは、ステークホルダ合意の重要性をまったく何もわかっていない。（DEOS コンサルタントの受け売りを繰り返す。）万が一、勝手に運用を変えて、データが消えてしまったら、それこそ、２次災害もいいところよ。

ちぇんじぃ　ごめんなさい。

みょーあん　(ミョーだな。D-ADD の導入は、いつ合意[1]したのだ？？)

ファイルサーバの非同期モードで運用すれば、万事解決だ。ちぇんじぃは極秘裏に問題解決を図る決心を固めた。

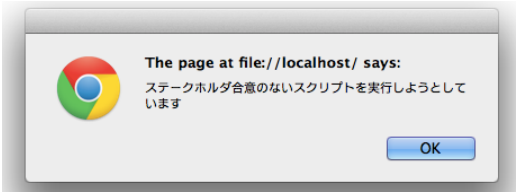


図５ スクリプトを実行したときの警告画面

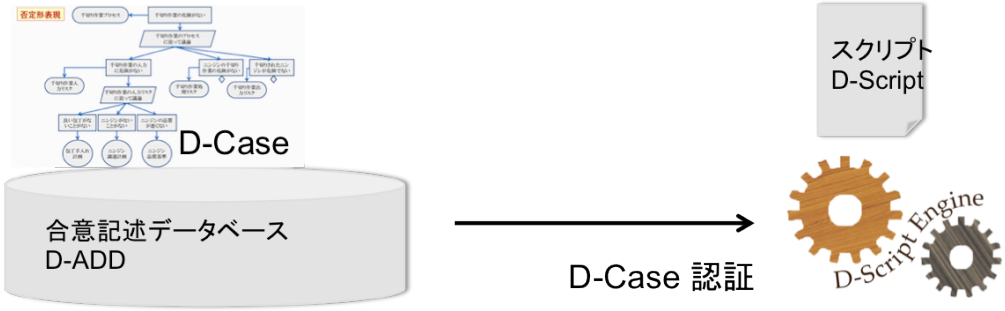


図６ D-ADD は、D-Case によるステークホルダ合意を格納したデータベースです。D-ADD と連携することで、合意のないスクリプトの実行が防がれます。

ドリーたちは、I/O ボトルネックを回避するため、メインメモリ 512G バイトマシンの RAM ディスクを使うことを考えた。みんなが納得する障害対応スクリプトは作れるのだろうか？

障害対応スクリプトを設計する

ドリー　もう一度、障害対策の手続きをまとめてみて。

みょーあん　「ピーク性能のときに、高性能マシンの RAM ディスクにデータをコピーして、ロードバランサーでマシンの稼働を切り替える。」

ドリー　なんかわかりにくいわね。

ちぇんじぃ　はい。DEOS にしたがって、設計図を書いてみました。

ドリー　なるほど。３つのタスクにわけて実行するわけね。

ちぇんじぃ　それぞれのタスクをスクリプトで書いていきます。

みょーあん　もし切り替え前のタスクに失敗したときはどうなるの？

ちぇんじぃ　D-Script のお作法では、スクリプトのアトミシティを保てるように設計するので、途中の実行ミスを検出したら、最初の状態に巻き戻してくれるよ。

みょーあん　なるほど、実行ミスした原因をとりのぞいて、もう一度、最初からスクリプトを実行すればいいわけねえ。

ドリー　タスクの失敗は、どれくらいの頻度で起こるの？

ちぇんじぃ　ほとんど起こらないと思いますが、根拠となるデータは示せません。

ドリー　それでは、せんせいに納得してもらって、合意は得るのは難しいかもね。

ちぇんじぃ　では、サーバ切り替えに失敗したときは、管理者だけでなく、せんせいにも事前に状況を通知し、せんせいの方でも授業の進め方等で工夫できる余地を作りましょう。

みょーあん　そこまでスクリプトに書く必要あるのかな？

ちぇんじぃ　これは、みょーあんが連絡を「おっと、うっかり」というのを防ぐ効果もあるんだよ。

スクリプトを図示することで、みんなで障害対応の議論を深めることができる

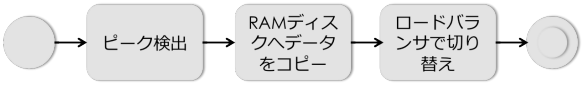


図７ D-Script の設計図

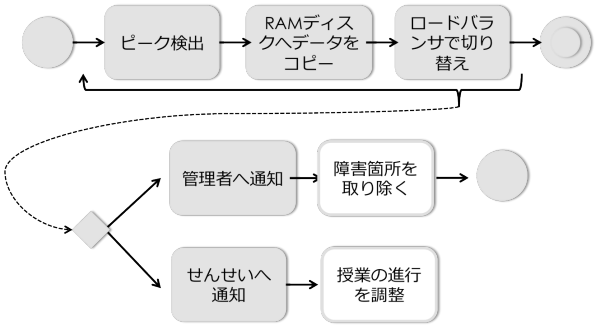


図８ ２次リスクを含めた D-Script の設計図

更なるステークホルダ合意に向けて

ドリーたちは、D-Script の設計図にもとづき、D-Script のテストを行い、同時に W 大の先生と最終合意をえるための、D-Case をまとめていた。

ドリー　W 大のせんせいから、そのサーバで十分だと言う根拠を見せてほしいと言われたわ。

みょーあん　Netperf で負荷テストしまおきました。40人同時アクセスで、リソース利用率は最大13%でした。

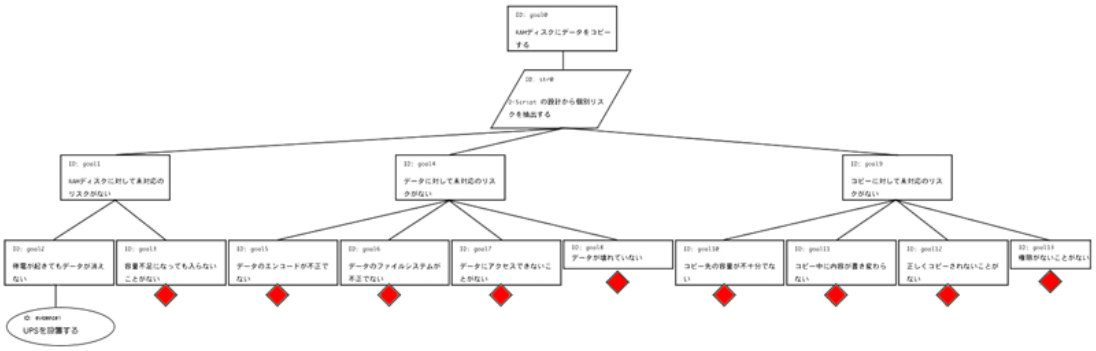
ドリー　それは、エビデンスとして D-Case に追加しよう。

ちぇんじぃ　「ピーク検出」の成功率が 50%前後なんだけど、これは大丈夫かな？

みょーあん　ビミョーだね。むしろ、大丈夫じゃないという根拠にもなるね。

D-Case
"RAMディスクにデータをコピーする"

図9 D-Case による D-Script のリスク議論



ドリー　ピーク検出は、（モニターの）サンプリングのタイミングによるから、危ないわね。

みょーあん　むしろ、過去ログをみれば、きれいに授業時間だけアクセスが集中しています。

ドリー　では、「ピーク時とは W 大演習時間のこととする」と限定しましょう。D-Case のコンテクストに追加して合意すれば、授業の時間が変更になったときは、連絡が期待できるわ。

みょーあん　D-Script の実行テストはどう？

ちぇんじぃ　負荷の低い状況では、正しく動くのを確認したよ。あとは、負荷の高い状況でも動いてくれることを祈るばかりだね。

ドリー　異常状態のテストはしなくていいの？

ちぇんじぃ　ここで保証すべきは、ソフトウェアバグ（プログラマに起因する失敗）に限定した方がいいと思います。何でも保証していたらキリがありません。

みょーあん　本当にそれでいいのかな…。

ドリー　そのドーリね。ちぇんじぃのいうように、何でも保証していたらキリがないわね。あとは、せんせいを納得させるための、私の説得術にかかっているわけね。任せてちょうだいよ。

せんせいは、D-Case をみながら、納得した顔でいった。今週の授業は、大丈夫そうですな。

5月中旬、木曜日4限。はじめて、1回の Proxy Error も表示されずに授業を終えることができた。どうやら、障害対応スクリプトは、うまく機能したようだ。

ドリーは、感謝半分、自慢半分という様子で、DEOS コンサルタントの匿名希望に電話をかけた。

ドリー　匿名希望さん、先日はありがとう。

匿名希望　うまく解決できましたか？

ドリー　ええ。それはおかげさまで。

匿名希望　それはよかったですね。ちょっと、D-Script と D-Case を拝見できますか？

ドリー　ええ。もちろん。

匿名希望　実は、ドリーさんには黙っていたのですが、D-Script にはリスク発見ツールがあって、ちょっとそれをかけてみると、

ドリー　ちょっと（怒）秘密にするなんてひどい！！

匿名希望　タダでは出せませんよ。えーと、D-Script の設計からリスクを発見し、それが D-Case のゴールに展開されると…むむむ。

ドリー　何か、問題は見つかったかしら？

匿名希望　「RAM ディスク」ですね。停電対策していますか？

ドリー　う？どうかしら、UPS で支えられる電源容量じゃないから…。

匿名希望　じゃ、授業中に停電になったら、データは消えてしまうね。

ドリー　ひゃー、大変！！

匿名希望　システム障害対策に、これで絶対に大丈夫ということはないよ。D-Case や D-Script は常に更新しないとねえ。

ドリー　ありがとう。じゃ。

匿名希望　D-Script リスク発見支援ツールの見積もりを送っておくよ。興味があるようだから。
（ようやく、元が取れそうだ。）

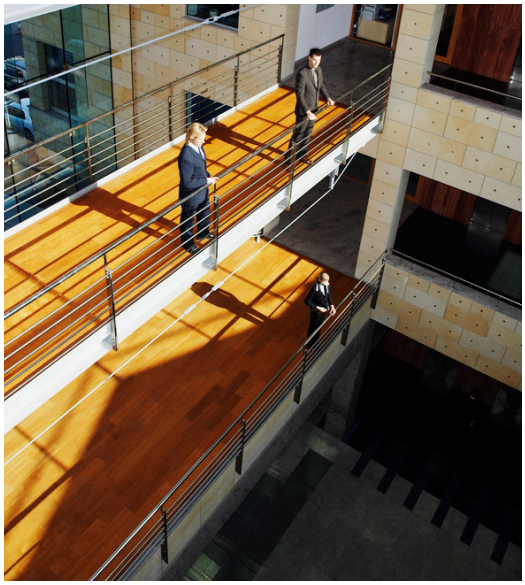
これで大丈夫と思わない

D-Case を書けば、それだけで安心ということはない。常に更新し続けることが大事だ

ドリー

ちぇんじぃ

みょーあん



DEOS コンサルタントによる技術解説

障害対応はどうやってスクリプトで書くべきなのか？

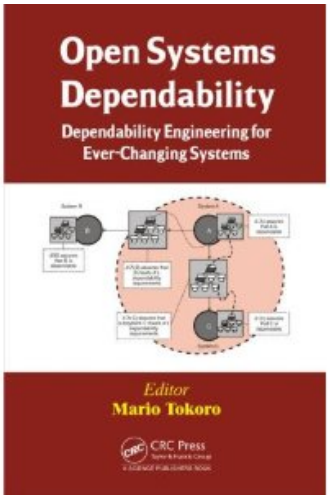


図１０ DEOS のバイブル

障害対応と D-Script の役割

D-Script プロジェクトは、スクリプトの柔軟さを活かして、DEOS プロセス[3]で規定する運用時のプロセスをサポートするように研究開発が進められました。

- 未然防止
- 迅速対応
- 原因究明
- （合意にもとづく）変化対応

もともと、スクリプト技術自体は、Bourne Shell (bash)のようにシステム運用を助けるために作られたものです。スクリプトは、システム運用の変更にあわせて、自由に書き換えられることから、現場のエンジニアリングで広く使われています。

その一方、ソフトウェア開発に比べると、スクリプトの記述は、リスク分析もソフトウェアテストも不十分というのが現状です。それがいろいろと問題となります。

まずは、ASPEN ケースをふりかえりながら、スクリプト技術の課題と今後の展望をみていきたいと思います。

未然防止：メンテナンスのスクリプト化

ASPEN のケースでは、サービス開始に先立ち、トラブルが発生しそうな箇所についてリスクを議論しました。ドリーさんのように、上司にあたる人がリスクに興味を示すことは、大変によいことです。

ハードウェアやシステム過負荷は、十分に予測して、適切なメンテナンスを実施することで、サービス全体への影響を軽減させることができます。また、W 大へのサービスが拡大したように、サービス環境が変わったら、それに応じてメンテナンス体制の見直しをすることも大変にいいことでしょう。

問題は、「定期的にバックアップをお願い」と依頼し、現場のエンジニアに任せっきりにした点です。このあと、みょーあん氏は、メンテナンス作業をスクリプト化したわけですが、ドリーさんから何もチェックも受けなかったようです。

図１１は、みょーあん氏から頂いたバックアップ･スクリプトです。もっとも、みょーあん氏が自分のスクリプトをドリーさんに見せたとしても、安全性をチェックするのは難しかったでしょう。

今回は、バックアップスクリプトが原因で大きな事故になりませんでしたが、もしバックアップスクリプトが動作しなかったら、大事故につながります。ちゃんと、スクリプトの記述と運用を管理する体制を作っていかないといけません。

障害発生のメカニズム

ドリーさんたちの障害対応ぶりを見ていく前に、そもそもどのように障害発生が発生するか、前提知識を入れておきましょう。

まず、ディペンダビリティの専門家の間では、障害発生のメカニズムをフォルト(fault)、エラー(error)、失敗/障害 (failure)の３段階[4]に区別して議論します。

フォルトは、障害要因となりうるイベント、もしくは行動です。エラーは、フォルトによって出現する正常状態から逸脱した状態(誤差)となります。エラーは放置すると、失敗、つまりサービスが提供できない事態につながります。また、ある失敗は、別の失敗を引き起こすフォルトとなり、図１２のようにエラー伝搬が説明されます。

フォルトと失敗/障害を分離するのは、重要な意味があります。一般に、日本語では「障害対策」と呼びますが、その結果、失敗/障害に着目して議論しがちです。「同じ障害を繰り返さない！」みたいなスローガンに違和感を覚えるのではないでしょうか？しかし、フォルトが違えば、障害発生の過程が異なるので、結果として同じ障害になることはあります。

ディペンダビリティの専門家は、結果よりもその原因となるフォルトに着目します。システム障害への対応手段として、フォルト予測(fault forecasting)、フォルト除去(fault removal)、フォルト寛容(fault tolerance)、フォルト回避(fault prevention)のように、すべてフォルトに着目して行

```
#!/usr/bin/dscript

# backup.dscript - backup user data by myoan


const SOURCE_DIR = "/var/lib/aspen/resource"

const DEST_DIR   = "/home/admin/aspen/backup"

boolean backup_userdata() {

  test -d "${DEST_DIR}/backup.0"

  if($? == 0) {

    rm -rf "${DEST_DIR}/backup.3"

    mv "${DEST_DIR}/backup.2" "${DEST_DIR}/backup.3"

    mv "${DEST_DIR}/backup.1" "${DEST_DIR}/backup.2"

    mv "${DEST_DIR}/backup.0" "${DEST_DIR}/backup.1"

  }

  rsync --archive --delete -F --link-dest=../backup.1 \

              "${SOURCE_DIR}" "${DEST_DIR}/backup.pre"

  if($? == 0) {

    mv "${DEST_DIR}/backup.pre" "${DEST_DIR}/backup.0"

    return true

  }

  return false

}

backup_userdata()
```

図１１ D-Script によるバックアップスクリプト

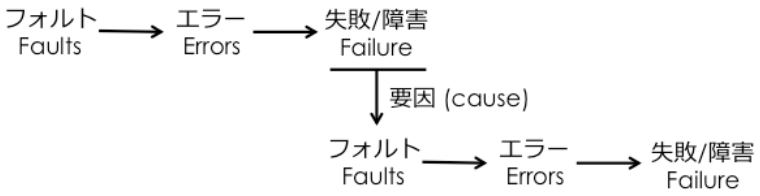


図１２ フォルト、エラー、失敗/障害のモデル

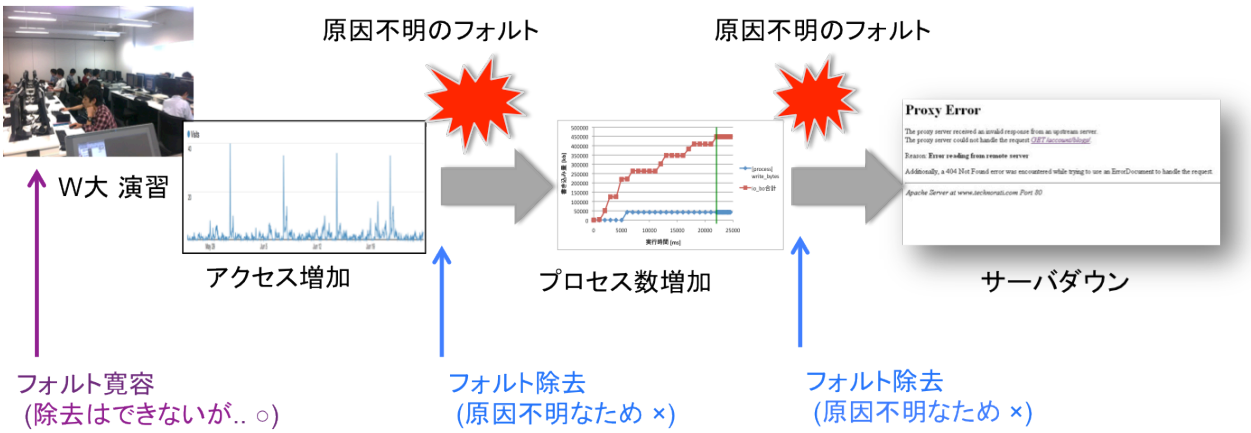


図１３ ASPEN のケースにおけるエラー伝搬

います。当然、障害対応もフォルトに着目して行うべきです。

もうひとつ注意したいのは、モニターシステムとフォルトの関係です。一般に、モニターで監視できるのは、CPU 利用率のようなシステムの状態、つまり正常とエラー(逸脱)を判定することができる値となります。フォルトを直接モニターすることはできません。たとえば、CPU 負荷が異常値を示しても、それを引き起こしたフォルトまではわかりません。フォルトの種類を特定するためには、過去のイベントを記録したログなどをあわせて解析する必要があります。

要因特定の難しさ

障害を正しく処置するのは、一にも二にもフォルト、つまり要因を正しく特定することです。間違った要因分析にもとづいて行動すると、障害が繰り返し再発するだけでなく、より悪い状態に陥ることもあります。

ASPEN のケースでは、W 大の授業中にアクセス数が増大し、そのときサーバ上でプロセスが大量に生成されることがわかりました。しかし、なぜアクセス数が増えると、プロセスが大量に生成されるのか、肝心のフォルトとなる根本要因はわかりません。

また、OS の方も不思議です。単にプロセスが大量に生成されても、プロセステーブルが溢れるだけで、OS 自体が落ちる理由もよくわかりません。

これをエラー伝搬モデルにまとめてみると、図１３のとおりになります。

障害対応として、最初に考えるのは、フォルトとなる要素を取り除くことです。しかし、まさか W 大からのアクセス数を減らすわけにはいきません。大量プロセスの生成の原因自体もわからないので、取り除くことはできません。もうひとつ加えていえば、大量プロセス生成はモニターで観測されただけで、それが本当にサーバダウンの引き金といえるのかわかりません。

障害対応に不慣れな素人は、往々にして、要因分析が不十分なまま障害対応を間違えてしまいます。しかし、もっと怖いのは、要因分析に時間を費やし過ぎて対応がはじめられない状態です。

今回、ドリーさんがプロセス大量生成の要因特定を深追いせず、サーバ増設を検討したのはナイス判断だったといえます。

仮にプロセス大量生成の原因がわかっても、ソフトウェアのバグに起因する可能性が高いので、短期間にフォルト除去はできません。それよりは、W 大からのアクセス増に対して、フォルト寛容（フォルトトレラント）なアプローチで障害対応を進めた方が確実というわけです。

ドリーさんがどこまで冷静に分析していたかわかりませんが、発生した障害に対応する必要があるときは、フォルト除去よりも、フォルト寛容化（多重化、バックアップ）を採用することがうまくいきます。障害対応のポイントは多くありますので、特に迅速対応が必要な場合は、時間のかからない手法を選択するのが重要になります。

迅速対応と自動応答の違い

迅速対応は、システム障害が発生しサービスが停止してしまったとき、影響や障害を最小限におさえるアクションです。今回のケースでは、せんせいから障害報告があり、みょーあん氏が「再起動する」という部分が迅速対応に相当します。しかし、みょーあん氏は、ASPEN のシステムダウンに当初気づかずに、ユーザからの指摘ではじめて気づき、対応を行うというゴテゴテの対応を演じてしまいました。

ここでひとつ疑問に感じませんか？ どうして、みょーあん氏は自動「再起動スクリプト」を用意し

なかったのでしょうか？もしモニターがサーバ異常を検出したら、自動的に再起動をかけるようにしておけば、少なくともせんせいから指摘されてから対応するという失態は避けられたはずです。

もしシステム運用者が、みょーあん氏ではなく、開発能力のあるちぇんじぃ氏であったら、モニターと再起動スクリプトを組み合わせて、自動応答システムを組み上げられたと思います。

しかし、自動応答システムには、障害対応の素人が陥りやすいワナがあります。

まず注意したいのは、一見、無害にみえる「再起動」であっても、より状況を悪くする可能性はあります。たとえば、サーバを不用意に再起動してしまったため、他のサーバにアクセスが集中し、連鎖的にサーバがダウンしたという事例も過去にありました。再起動してしまったため、システム状態が不明になって要因解析の根拠が消えることもあります。

このように、再起動の対策ひとつとっても、安全に再起動できるタイミングはいつか？再起動することによる影響はないか？様々な要素を検討する必要があります。

しかし、コンピュータシステムの障害発生メカニズムは、それほど因果関係がはっきりしません。エアコンの温度コントロールシステムのように決定的に動作させることはできないのです。

スクリプトは、何でも書けるからといって、障害対策の自動応答システムを構築するのは危険です。唯一、どんな場合でも使える迅速対応は、システム運用者に連絡を入れることです。もちろん、大抵のシステム運用者は、呼び出されることを嫌いますが…。

変化対応への合意

ASPEN のケースで、最も興味深い点は、当初計画していた「サーバを増設する」という対応がうまくいかず、「高性能マシンの RAM ディスクで対応する」という措置に切り替えた点です。これは、最初の想定と異なるために、一種の変化対応といえます。このとき、ステークホルダの間でリスクを議論し、合意形成を行うことが必要です。

ドリーさんたちは、D-Case ツールの助けを借りて議論をまとめることができました。さらに、より大きなプロジェクトになれば、D-ADD 合意形成システムのようなツール支援も必要になってくるでしょう。 DEOS プロジェクトでは、このような支援ツールをいくつも開発、提供しています。

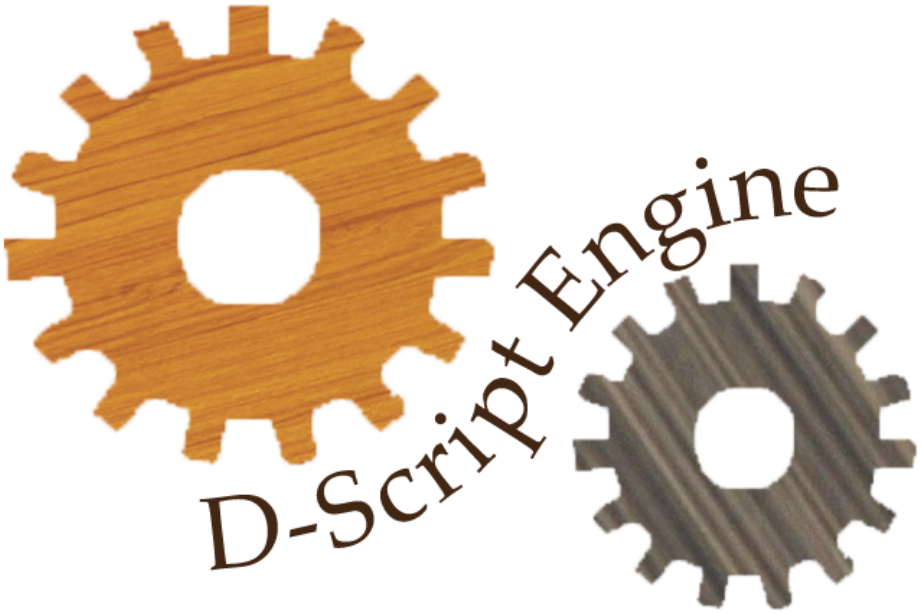
しかし、ここで注意したいポイントが２つあります。

- リスクの網羅性をどうやってあげるか？
- どこまでエビデンス（根拠）を提示するか？

前者は、問題発見から解決まで期間が短いとき、リスクの洗い出しが不十分になりがちです。実際、ドリーさんたちは、停電への備えなしに、RAM ディスクでデータ処理をしてしまいました。もし停電と W 大の授業が重なれば、データが消えてしまったでしょう。大きなシステム障害は、不幸が重なったときに発生するものです。そもそも、スクリプトの設計自体に問題があったというのは、リスク分析の段階で避けたいものです。

また、後者のポイントは、みょーあん氏とちぇんじぃ氏の間で、障害対応スクリプトをどこまでテストするかという点で論争になりました。みょーあん氏のいうとおり、異常負荷の状態で動作するスクリプトなので、異常状態でもテストすべきでしょうか？それとも、正常状態のテストだけで十分なのでしょうか？

ここで重要になってくるのは、スクリプト処理系がどこまで保証し、スクリプトの記述者が何を保証しなければならないか、明確になっている点です。もちろん、スクリプト処理系が多くのことを保証してくれれば、スクリプトの記述者の負担は軽くなります。



現在のコンピュータシステムは、メンテナンスや緊急時のバックアップなど、スクリプトによる支援なしでは運用できません。同時に、そのようなスクリプトは、運用システムのディペンダビリティ、つまり保守や障害対応などに密接に関連した部分で利用されるため、一旦、スクリプトが正常に動作しなかった場合は、システム障害が深刻化します。

D-Script プロジェクトは、スクリプト実行に失敗するケースを想定し、それらの失敗にどうやって対応可能するか、その対応をツール支援するという形で、新しいスクリプト言語処理系 D-Script Engine を設計しています。

設計思想：フォルトマトリックス

D-Script の基本構想となるのが、現場のエンジニア視点でまとめた、D-Script フォルトマトリックスです。スクリプトの実行中に発生するエラーに対し、そのフォルトを誰が除去できるか、という視点から４つの分類をしてあります。

ソフトウェアフォルト(Software Fault) - 対象とするプログラムが開発段階で混入することが要因で発生するフォルト、設計ミスやバグなど。フォルト除去には開発プロセスでの修正が必要であり、オンラインでのフォルト除去は不可能。

ユーザフォルト (User Fault) – ユーザの入力ミスや入力されたデータの不整合（エラー状態）を要因として発生するフォルト。入力データを修正することで、フォルトを除去できる。

システムフォルト (System Fault) – 言語ランタイムが動作するコンピュータシステムが、何らかの原因で通常とは異なる動作を行い、それがもとでアプリケーションのエラー状態が引き起こされるフォルト。システム状態を正常状態（初期化、バックアップ）に戻すことで、フォルトを除去できる。

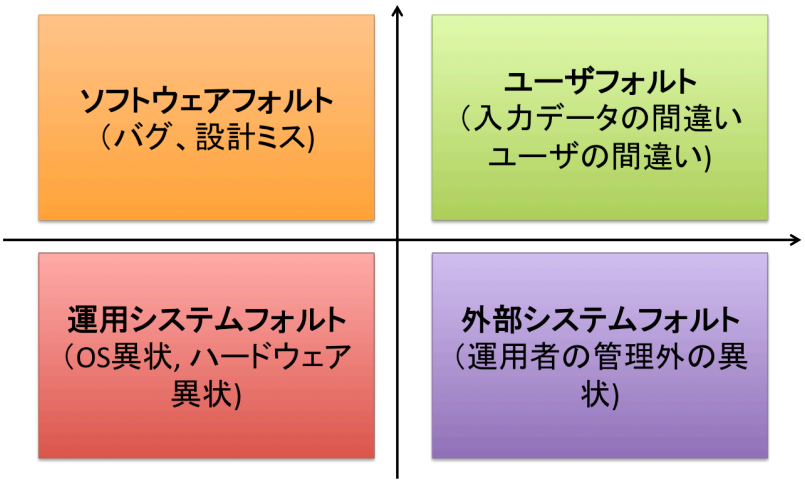


図１４ D-Script フォルトマトリックス

外部フォルト (External Fault) – ネットワークや外部サービスなど、外部の管理ドメインのエラー状態によって発生するフォルト。フォルトを除去するためには、管理ドメインのエラー回復が必要である。

フォルト診断機能の重要性

D-Script Engine の特徴は、このフォルトマトリックスにもとづいて、エラーのフォルト診断を行う機能を備えている点です。ASPEN のケースでもあきらかのとおり、エラー状態は簡単にわかるのですが、そのフォルトはなかなか特定するのは難しいです。フォルト診断は、フォルトの特定は難しいにしても、誰が対応すべきフォルトなのか、原因を切り分けることを目指しています。

もう少し、具体的にフォルト診断の例をみていきましょう。これは、２年前の ET2010 展示会であった実例ですが、あるスクリプトが展示会場に持ってきたら動かなくなりました。

展示会場の担当者は、スクリプトが動かないということで大慌てになりました。結局、展示会場の担当者にとって、エラーメッセージが原因不明だったため、スクリプトのバグと判断され、開発者が呼び出されました。早朝、スクリプト開発者がチェックしてみると、結局、データベースサーバが起動していないだけでした。

データベースが起動していないだけで、展示会初日の午前中を失いました。スクリプトは、いろいろな理由で動かなくなるため、原因の切り分けを行うだけで、貴重な時間を失わなくて済みます。

D-Script フォルト診断機能

D-Script Engine は、言語処理系内部の言語処理情報、定期的なシステム情報、エラーメッセージの診断を組み合わせて、フォルト診断を行います。

次は、”/etc/passwd” に書き込

図１５ フォルト診断の難しさ

(出典 http://www.feldstudie.net/2012/04/19/picdump-19-04-12-der-nerd-picdump)

```
IOException: UserFault SystemFault

StackTrace

[8] ((shell):4) System.fopen(filename=(String)
"/etc/passwd", mode=(String) "w")



SYSLOG {"LogPoint": "ErrorPoint", "FaultType":
"SystemFault,UserFault", "ScriptName":
"initiate", "ScriptLine": 4, "Api": "fopen",
"filename": "/etc/passwd", "mode": "w", "Errno":
13, "Message": "Permission denied"}
```

図１６ D-Script のログ出力とフォルト診断の例

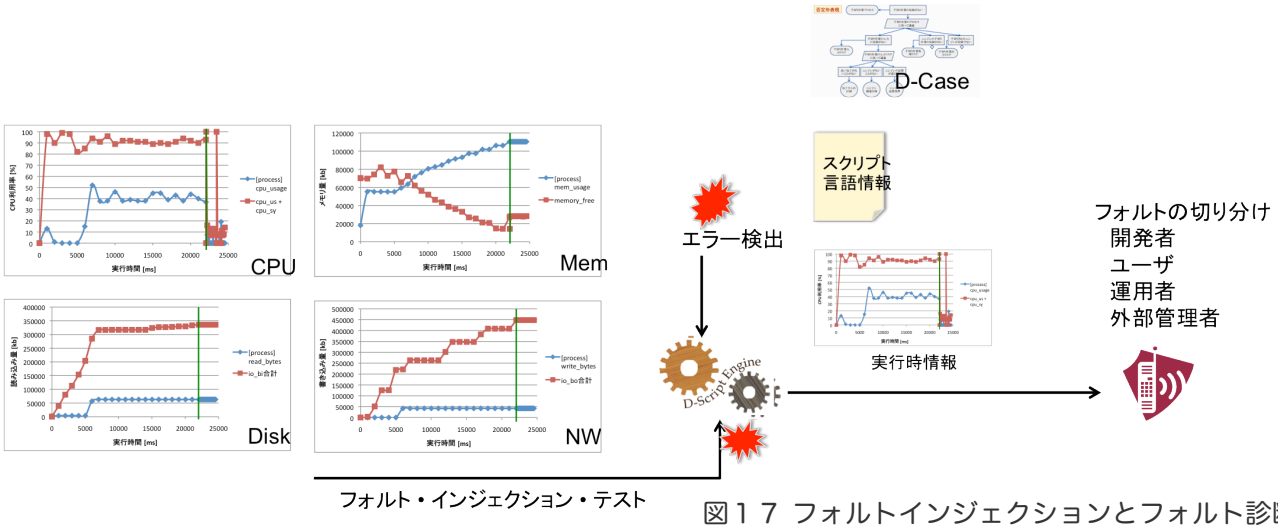


図１７ フォルトインジェクションとフォルト診断

もうとしたときのエラーとフォルト診断の表示例です。スタックトレースを表示するだけでなく、フォルトの種類を分類します。

D-Script Engine は、システムフォルトの診断を行うため、フォルトインジェクション機能を用いて、あらかじめ様々な不安定な動作環境での動作検証を行っています。図は、様々な高負荷環境を起こしたときの D-Script Engine のプロファイルの例です。このようなフォルトインジェクション試験を繰り返し、データとして蓄積することで、より正確なフォルト診断を目指しています。

D-Case/D-Script の連携

D-Script Engine のフォルト診断を実現する上での最大のポイントは、ソフトウェアフォルト、つまりバグや論理的な設計ミスの診断にあります。そもそも、プログラミング言語処理系が自分自身で診断できるものでしょうか？

たとえば、無限ループという分かりやすいバグがあります。しかし、それがプログラマが意図した通りなのか、それとも、バグなのか機械的に診断することは難しいです。

そこで重要になってくるのが、D-Case[5]との連携となります。D-Script のリスク分析から設計、ソフトウェアテストの結果まで、D-Case で議論し、その結果は記録されます。D-Script Engine は、D-Case 情報をオンラインで活用することで、実行前の D-Case のエビデンスと D-Script Engine 内部の診断情報からフォルト識別の精度をあげます。

逆の視点からみると、D-Script のフォルトマトリックスは、D-Script がカバーできる範囲はどこまでかをより明確化できるため、D-Case に何を書く必要があるか具体化するのに役立ちます。

D-Case 認証機構は、D-Script Engine のもっともアドバンスなセキュリティ機能です。D-Case のエビデンスが十分でない場合は、D-Script の実行自体を停止させることができます。もちろん、どの程度、ソフトウェアフォルトのリスクがないかは、D-Case における合意レベルによります。

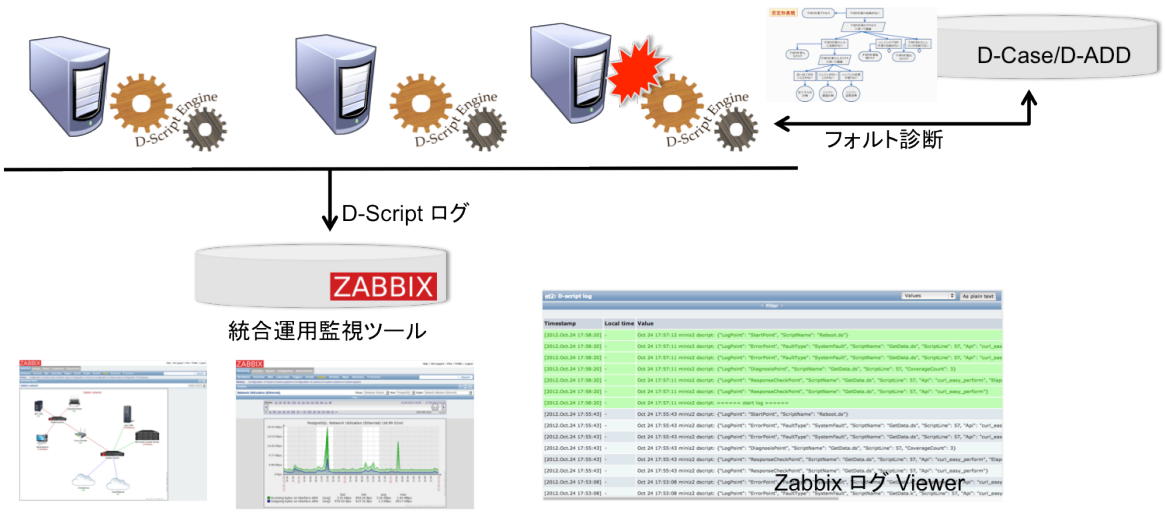


図１８ Zabbix と D-Script との連携

スクリプトは、システム運用の広範囲にわたり、ソフトウェアシステム間のハブとして役割を果たすため、D-Case 認証機構により、システム運用と D-Case の連携がより確実になります。

運用システムの連携

D-Script Engine は、D-Case 認証機構を組み合わせて、システム運用のハブ、複数のソフトウェアシステムを連携させる重要なパートとして機能します。

D-Script Engine は、自分自身が検出したエラーのフォルト診断以外にも、将来、エラーとなりそうなポイントにおいて自動的にログを記録します。

D-Script Engine は、次のようなログポイントをサポートしています。

- ErrorPoint – エラーを検出したとき
- SystemChangePoint -- システムに永続的な変更を与えたとき
- ResponseCheckPoint – サブコンポーネントの実行時間が予想より長かったとき
- SecurityAudit -- セキュリティ監査上、記録しておくべき情報

D-Script のログは、既存の統合運用監視ツールと互換性を持つように設計されています。図は、Zabbix オープンソース監視システムから、D-Script の挙動を監視する連携の例です。このように、システム間の連携において、D-Script Engine を用いれば、より詳細なフォルト情報を得ながら、システム運用が可能になります。

オープンソース実装

D-Script Engine は、静的スクリプト言語 Konoha を最小化し、処理系の実装に対してバグ検証を行った Mini Konoha をベースに開発されました。本冊子で紹介したフォルト診断機能、D-Case との連携機能などは、完全にモジュール化され、動作環境にあわせてカスタマイズすることが可能です。

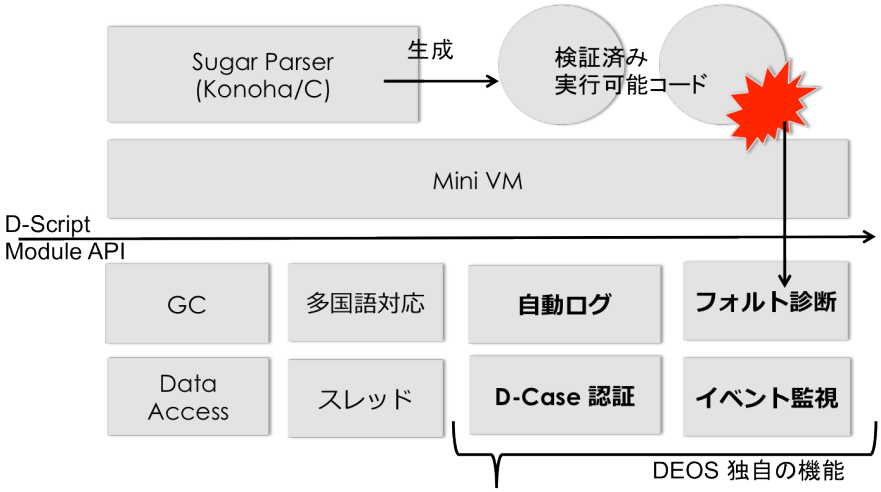


図１９ D-Script Engine のアーキテクチャ

D-Script の文法は、C スタイルの最小限の文法＋シェル言語で使われるコマンドとパイプを採用しています。システム管理で使われるシェルプログラミングの知識や経験を活かして書くことができます。

D-Script 言語は、Mini Konoha をベースにしているため、Mini Konoha の提供する優れた文法拡張機構を用いることができます。D-Case データとの連携したスクリプトから、クラウド環境における VMM 制御まで、様々なスクリプティングが可能になります。D-Script のプログラミングは、紙面の都合により、参考文献[6]などを参照してください。

D-Script Engine は、DEOS プロジェクトの公式ページからオープンソース公開される予定です。現在は、開発コミュニティである Mini Konoha プロジェクトのホームページ[7]から参照実装をえることができます。■

参考文献

[1] 株式売買システムの障害発生に関する再発防止措置等について http://www.tse.or.jp/news/03/120216_a.html

[2] 調査報告書 （最終報告書） http://support.fsv.jp/urgent/pdf/fs-report.pdf

[3] Mario Tokoro (Ed.). Open Systems Dependability: Dependability Engineering for Ever-Changing Systems. CRC Press, 2012.

[4] Avizienis, A.; Laprie, J.-C.; Randell, B.; Landwehr, C.; , "Basic concepts and taxonomy of dependable and secure computing," Dependable and Secure Computing, IEEE Transactions on , vol.1, no.1, pp. 11- 33, Jan.-March 2004

[5] 松野裕，高井利憲，山本修一郎,他 D-Case 入門. ディペンダビリティケースを書いてみよう.

[6] 倉光君郎. Mini Konoha: 高信頼なスクリプトの実現に向けて. (近日刊予定)

[7] http://github.com/konoha-project


Dependable Embedded Operating Systems • Dependability Engineering for Open Systems •
DEOS