DEOS-FY2013-RE-01J

JST-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」

DEOS プロジェクト

D-RE (DEOS Runtime Environment) 仕様書

＜付＞ D-RE 導入ガイド

Version 1.0

2013/05/01

DEOS 研究開発センター

科学技術振興機構
Japan Science and Technology Agency

(空白ページ)

目次

1. D-RE 概要 4
2. 求められるシステム動作 5
2-1. アプリケーション監視（Application Monitoring） 5
2-2. システム監視（System Monitoring） 5
2-3. 証拠記録（Evidence Logging） 5
2-4. 分離（Isolation） 6
2-5. リソース制限（Quota） 6
2-6. 取り消し（Undo） 6
2-7. 移動（Migration） 6
3. D-RE 構成要素の設計思想と内部構造 7
3-1. D-Visor 7
3-2. D-System Monitor 7
3-3. D-Application Manager 8
3-4. D-Application Monitor 9
3-5. D-Box 11
3-6. D-Script Engine 13
4. D-RE の使用方法 13
4-1. D-Visor の使用方法 14
4-2. D-System Monitor の使用方法 14
4-3. D-Application Manager の使用方法 14
4-3-1. コマンド 14
4-3-2. ライブラリ 15
4-4. D-Application Monitor の使用方法 16
4-5. D-Box の使用方法 16
4-6. D-Script Engine の使用方法 16
5. おわりに 16
Appendix 17
A1. D-RE 導入ガイド 17
A1-1. 動作環境 17
A1-2. パッケージの入手 17
A1-3. インストール手順 17
A1-4. 初期設定 18

1. D-RE 概要

オープンシステムディペンダビリティ（OSD）の実現、すなわち「変化しつづけるシステムのサービス継続と説明責任の全う」のためには、以下が必要である。

- 継続的な改良改善のための開発と運用を連携する反復的なプロセス（DEOS プロセス）
- DEOS プロセスを仕組みとして支える合意形成・確認手法・ツール（D-Case）と実行環境（D-RE: DEOS Runtime Environment）
- D-Case に基づいて、OSD 実現を具体化した、スクリプト（D-Script）およびアプリケーション・プログラム（D-Aware Application Program）

本 D-RE 仕様書では、この 2 番目の項目の内、特に DEOS 実行環境（D-RE:DEOS Runtime Environment）について記述する。この第 1 章では、OSD の実現のために求められる機能について一般的に記述し、第 2 章以降で、実際に開発した機能やその実装について記述する。

DEOS 実行環境（D-RE）はステークホルダ合意に基づくディペンダビリティを実現するサービスを提供するための実行環境であり、DEOS アーキテクチャ図（図 1）では右下の部分に表示されている。図に示されるように、D-RE は様々なサブシステムから構成される。

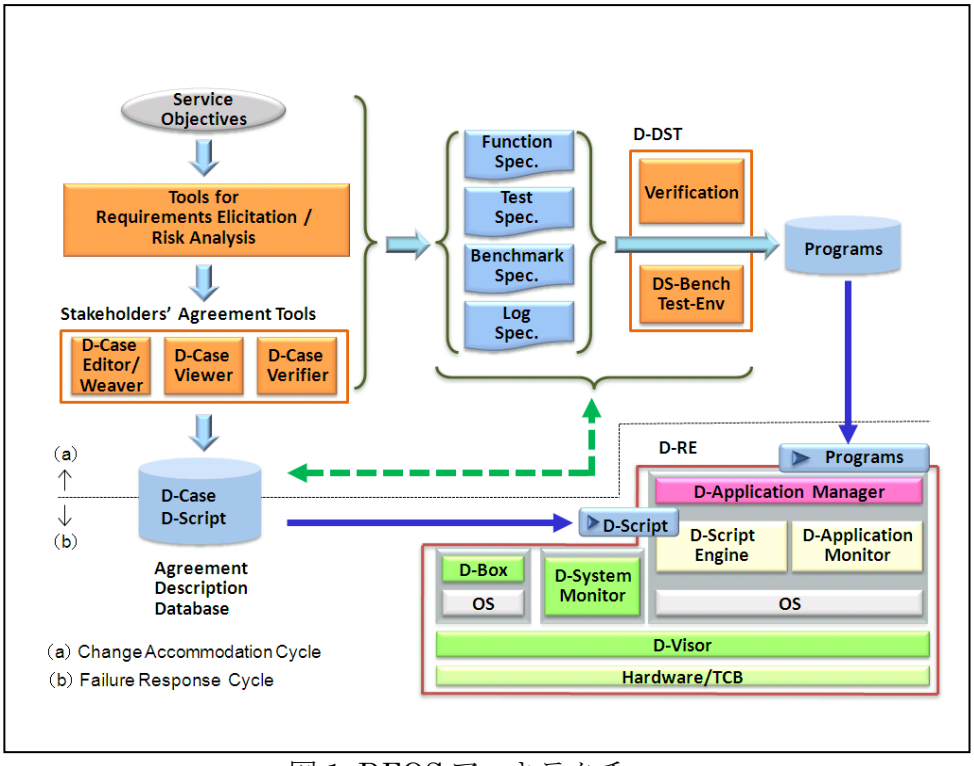

図 1. DEOS アーキテクチャ

D-Visor は、システムの構成要素の各々の独立性を担保する仕組み（System Container）を提供する。System Container は仮想マシン（VM）であり、個々に別々の OS が動いており、異なる System Container 内における異常や障害が他の System Container に波及することはない。D-Visor は図 1 のようにハードウェア上で直接動作する場合もあるが、Linux KVM を使用して System Container を実現する場合のように、ハードウェア上で OS が稼働し、その上で D-Visor が稼働する場合もある。

D-System Monitor は System Container に対する監視機能を提供する。VM を外側から観察し、コンテナ内で稼働する OS に影響を与えることなく、OS の改竄等による異常な振舞いを監視する。

D-Application Manager はアプリケーションの独立性を担保する仕組みとしてコンテナを提供する。コンテナとして System Container と Application Container があり、System Container は D-Visor によって提供される VM であり、Application Container は VM ではなく OS 上の軽量コンテナである。コンテナ毎に CPU やメモリ等のリソースを制限することができる。

D-Application Monitor はアプリケーションの動作監視機能を提供し、エビデンスを収集し、D-Box に蓄積する。

D-Box はエビデンスを始め、OSD 実現に有益な情報を安全・確実に記録する。

D-Script Engine は D-Script を安全・確実に実行する役割を担っている。D-Script には、何を監視しどのように対処するか記述してあるので、それによって D-Application Manager、D-Application Monitor、D-System Monitor を制御する。

2. 求められるシステム動作

ディペンダブルなシステムを実現するためには、通常運用や障害対応で様々なシステム動作が必要となる。ディペンダブルなシステムで求められる以下のような動作を実現するために、D-RE はベーシックな機能を提供している。

2-1. アプリケーション監視（Application Monitoring）

システム提供者はそれぞれの提供するアプリケーションに合った障害予知あるいは障害検知のために、何をモニタ／検出し、検出された結果どうアクションするのかを D-Script として記述する。モニタ／検出すべきデータはアプリケーション毎に異なるが、D-RE では一般的なモニタ項目として、コンテナ（System Container と Application Container）の CPU 使用率、メモリ使用量を取得できる仕組みを用意しているので、アプリケーションをコンテナ内で動作させることで、間接的に CPU 使用率やメモリ使用量を監視することができる。

2-2. システム監視（System Monitoring）

D-RE の提供する System Container は仮想マシン（VM）である。その死活や CPU 使用率、メモリ使用量を監視することによって、VM およびその上で稼働している Guest OS が正常動作しているかどうかある程度判断できる。Guest OS 上で個々の機能が実際に正常動作しているかどうかを判断するには、システム監視用のコマンドや監視ツール等を Guest OS 上で使用する必要がある。

また、D-RE で提供する D-System Monitor は、VM を外側から観察し、VM 上で稼働する Guest OS に影響を与えることなく、Guest OS の改竄等による異常な振舞いを監視することができる。VM を制御する Hypervisor（3 章で説明する D-Visor86 や Linux KVM）中に D-System Monitor をサポートするための機能を組み込むことにより、仮想デバイスの I/O データの取得・挿入や、Guest OS のメモリ内容の観察が D-System Monitor から実行できるようになっている。

2-3. 証拠記録（Evidence Logging）

D-RE には D-Box と言う記録機構がある。D-Box ではその機構への読み書き（アクセス）が厳重に管理され認可されたプログラムだけが実行できる。また記憶されるデータのダイジェスト（MD5）は RSA アルゴリズムで暗号化され、簡単には改竄できない。万が一データが改竄された場合は、改竄されたと言う事実が報告される。D-Box にはどのようなデータも記録することができるが、記憶容量には限りがあり、必要最小限の意味のあるものを記録するようにしないといわゆるログ破産を起こす。

D-Box に格納する・しないに関わらず、各アプリケーションが出力するログは、障害発生時の原因解析・原因説明に重要である。また、ログの出力フォーマットが統一されていると原因解析作業が効率よく実行できるだけでなく、パターンマッチングによる自動解析の可能性も拡大する。D-RE では、ログ出力を容易にするための関数群も提供している。

2-4. 分離（Isolation）

システムを開発する際には、一部の障害が全体に影響を及ぼすことが無く、一部の部分だけの分離が容易となるような全体構造を検討する必要がある。障害を起こした部分を速やかにシステムから切り離し（分離し）、その影響をシステムの他の部位に与えないで、その他のサービスを継続させることが、オープンシステムディペンダビリティの障害対応として重要な機能だからである。

D-RE では、System Container で OS レベルの起動・停止を実現し、Application Container ではプロセスのグループの起動・停止の機能を実現している。System Container は仮想マシン（VM）であり、それぞれのコンテナでは別々の Guest OS が稼働するので、1 つの VM 上の Guest OS に障害が発生して起動・停止を行なっても、他のコンテナ上の Guest OS には影響を与えない。また、Application Container は Linux Containers (LXC)を利用した軽量コンテナであり、1 つのコンテナ内のプロセス群と他のコンテナ内のプロセス群はお互いに影響を与えない。

2-5. リソース制限（Quota）

D-RE のコンテナでは、コンテナ内のプログラムが CPU を使う割合（CPU 使用率）とメモリを使う量（メモリ使用量）を、他のコンテナに影響を与えずに、制限することができる。この機能を使うことで、一部のコンテナ中のプログラムが暴走して CPU を占有し他のコンテナ中のプログラムに十分な CPU が割り当てられない状況や、一部のコンテナによるメモリの大量使用によってシステム内のメモリが枯渇してしまう状況を防止できる。D-RE によるリソース制限はコンテナ起動後に動的に設定できるため、システムやアプリケーションの監視の結果によってリソースを制限することで、障害に至らないで、サービスが遅れながらも継続できるようにするために使われる。

2-6. 取り消し（Undo）

システムに新しいサービスを追加したり、システムの不具合を直すために修正コードを追加したりする事はまれではない。ところがどんなに事前にテストを繰り返して来た追加コードや修正コードであっても、100%完全に予想通り動くと言う事はない。最悪の場合、システムダウンを引き起こし、ユーザーへのサービスを継続できない事態になる。このような場合、この作業を始める前の状態に戻せば、少なくともそれまでと同じサービスは提供できるので、取り消し（Undo）は重要な意味を持つ。

一度実行した処理を逆の順番にたどりながら個々に取り消していく操作は、一般的には容易ではない。そこで、処理開始前の時点の環境（データ）を保存しておき、その保存していた環境に戻すことで取り消し操作を実現することがよく行なわれる。D-RE でも、System Container、Application Container のそれぞれで、ある時点の環境を保存し、その環境から再開させる機能を提供している。

コンテナの環境を保存する際には、コンテナを完全に停止させた状態で行なう場合と、コンテナを一時的に停止させた状態で行なう場合がある。コンテナを完全に停止させた場合には、コンテナ内のプログラムはすべて終了しており、再開は通常のプログラムの開始と同じであるが、コンテナを一時的に停止させた場合には、CPU 割り当てが一時的に中断されているだけで、コンテナ内のプログラムは動作中でメモリも保持したままであり、再開は保持しているメモリを使用して一時停止中のプログラムの処理を続行することになる。D-RE では、前者の場合の保存・再開を snapshot の save/load 機能と呼び、後者を checkpoint/restart 機能と呼んでいる。

2-7. 移動（Migration）

D-RE では、System Container 上の Guest OS を他の System Container にコンテナ単位で移動させることができる。いわゆる VM の migration 機能である。同一ホストマシン上での移動はもちろん、ネットワークに接続された物理的に別のマシンとの間でもコンテナ単位で移動させる事ができる。ネット

ワークで接続された別マシンへ移動させる場合には、移動先のマシン上でも System Container が利用可能となっている必要がある。System Container 上の Guest OS を一旦停止してから移動させることだけでなく、Guest OS を稼働させたまま、移動させること、いわゆる live migration も可能である。ハードウェアに起因する異常を回避するためと言うより、そのハードウェアを入れ替えたり、修理したりする時に、システムを止めることなくサービスを継続するために使うシステム動作である。

3. D-RE 構成要素の設計思想と内部構造

D-RE は、ディペンダブルなオープンシステムを構築するための枠組みとなるベーシック機能を提供しており、以下のようなサブシステムから構成されている。

3-1. D-Visor

D-Visor は、システムの構成要素の各々の独立性を担保する仕組み（System Container）を提供する仮想マシンモニタである。個々の System Container は仮想マシン（VM）であり、それぞれ独立した Guest OS が動いており、1 つの System Container 内における異常や障害が他の System Container に波及することはない。

筑波大学が開発した D-Visor86 は、マルチコアプロセッサ用の仮想マシンモニタである。マルチコア（Hyper Threading を含む）の x86 CPU 上で、プロセッサコア毎に修正を施した Linux カーネルを動作させることができる。1 つのプロセッサコア上の Linux で D-System Monitor を稼働させることにより、他のプロセッサコア上の Linux を監視することができる。

D-Application Manager の API やコマンドは、Linux KVM を D-Visor として使用している。各 System Container ではそれぞれ別の Guest OS が稼働しており、1 つの System Container 上の Guest OS の起動・終了等は他の System Container に影響を与えない。System Container 毎に異なる Guest OS を稼働させることも可能なので、独立性が高く依存関係がないアプリケーションは異なる System Container 上で動かすことで、障害が発生した場合の影響を抑えることができる。

3-2. D-System Monitor

D-System Monitor は D-Visor と連携して System Container に対する監視機能を提供する。VM を外側から観察し、コンテナ内で稼働する Guest OS に影響を与えることなく、Guest OS の改竄等による異常な振舞いを監視する。

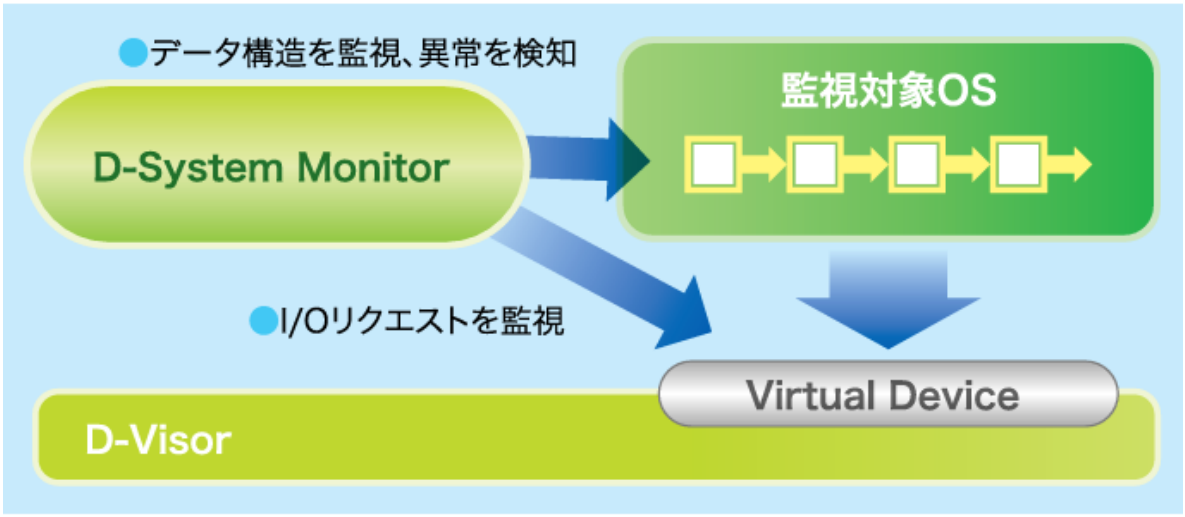

図 2. D-System Monitor

D-Visor は、仮想マシンの I/O リクエストを監視する機能と、監視対象 OS のメモリを直接参照する機能を D-System Monitor に提供する。D-System Monitor 上で動く監視機構では、それらの機能から得られるデータを利用し、監視対象 OS の異常な振舞いを検出する。現在のところ、早稲田大学と慶應義塾大学が開発した以下の 3 種類の監視機構が用意されている。

- FoxyKBD：キー入力の際の OS の挙動を監視する。大量のキー入力を疑似的に発生させ、その時に発生する I/O リクエストとの関連から、キー入力を横取りするマルウェアを検出する。
- RootkitLibra：ファイルの隠ぺいを監視する。NFS マウントしたディレクトリ上のファイルについて、システム上で見えているファイルと NFS パケット内のファイルの情報を比較し、ファイルの隠ぺいやファイルサイズの改竄を検出する。
- Waseda LMS：プロセスの隠ぺいを監視する。監視対象 OS のメモリを直接参照し、kernel データであるプロセスリストと実行キューのデータを比較することで、プロセスの隠ぺいを検出する。

これらの監視機構はいずれも、筑波大学が開発した D-Visor86 上で稼働する D-System Monitor で利用可能である。また、D-System Monitor を組み込んだ QEMU-KVM も開発しており、KVM の System Container に対して上記 3 種類の監視機構が利用可能である。

3-3. D-Application Manager

D-Application Manager は、アプリケーションの独立性を担保する仕組みとしてコンテナを提供する。コンテナとしては System Container と Application Container がある。また、D-Application Manager は、アプリケーションプログラムのログ出力や終了処理を支援するための関数群もライブラリとして提供する。

System Container は D-Visor によって提供される仮想マシン（VM）である。前述したように、D-Visor として D-Visor86 と Linux KVM が使用できるが、D-Application Manager は、API やコマンド経由で提供している System Container として、Linux KVM の VM を使用している。Linux KVM を D-Visor として使用すると、ハードウェア上で OS（Linux）が稼働し、その OS 上で D-Visor が動作することになるため、D-Visor の制御を同じ OS 上の D-Application Manager の API やコマンドとして提供できる。

個々の System Container では、それぞれ独立した Guest OS が動いており、1 つの System Container 内における異常や障害が他の System Container に波及することはない。また、System Container 毎に CPU 使用率やメモリ使用量を制限することが可能であり、リソースが枯渇しないようにシステム全体で適切な配分を行なうことができる。

System Container については checkpoint/restart 機能と snapshot の save/load 機能を共に提供している。checkpoint/restart は VM を一時停止した状態でデータを保存する。CPU 割り当てが一時的に中止されているだけで、コンテナ内の Guest OS やその上で動くプロセスは動作が終了しておらず、メモ

リ中のデータを保持したままであり、再開時にはこの一時停止中の Guest OS やプロセスは、その途中まで実行している処理を続行することになる。一方、snapshot の save/load 機能は VM 上の OS が終了した状態のファイルシステムのデータを保存するので、再開は通常の OS の起動と同じである。checkpoint/restart 機能あるいは snapshot の save/load 機能を使って更新プログラムの導入やデータ変更の前の状態を保存しておくことで、障害発生時には Undo して元の状態に戻すことができる。

さらに、System Container では migration 機能も提供している。System Container 上の Guest OS を他の System Container にコンテナ単位で移動させる、いわゆる VM の migration 機能である。同一ホストマシン上での移動はもちろん、ネットワークに接続された物理的に別のマシンとの間でもコンテナ単位で移動させる事ができる。System Container 上の Guest OS を一旦停止してから移動させることだけでなく、Guest OS を稼働させたまま移動させる、いわゆる live migration も可能である。

System Container の生成の際に KVM イメージを使用するが、そのベースとなるイメージ（以下「ベースイメージ」）は、D-RE が用意しているコマンドで作成可能である。代わりにユーザーが用意した KVM イメージを使用して System Container を生成することも可能である。

Application Container は VM ではなく OS 上の軽量コンテナである。D-Application Manager では、LXC のコンテナを API やコマンド経由で Application Container として提供している。それぞれのコンテナは 1 つの OS 上で動作しているため OS の障害は全コンテナに影響してしまうが、コンテナ内のプロセスは他のコンテナ内のプロセスから独立しており影響を受けない。Application Container によって複数のプロセスを 1 つのグループとして扱うことができ、そのグループ毎に CPU 使用率やメモリ使用量を制限することができる。同一 OS 上で稼働する方が都合が良いアプリケーションも、異なる Application Container 上で動かすことにより、CPU やメモリ等のリソースを分離することができ、障害発生時の他のアプリケーションへの影響を少なくすることができる。

Application Container でも、snapshot の save/load 機能が提供されている。しかし、LXC の checkpoint/restart 機能が未実装のため、Application Container の checkpoint/restart 機能は D-RE でも提供されない。

System Container と Application Container は共に、生成(create)→開始(start)→停止(stop)→破棄(destroy)という手順で使用される。その指示はコマンドや関数から実行でき、dre-sys、dre-app コマンドや libdresys、libdreapp ライブラリとして提供している。また、checkpoint/restart 機能や snapshot の save/load 機能もそれぞれのコマンドやライブラリから使用可能である。

D-Application Manager は、アプリケーションプログラムを作成する際に利用できる libdaware ライブラリとして、ログ出力や終了処理を支援する機能を提供している。ログ出力のためには、syslog へのログ出力のフォーマットを簡単に指定するための関数やマクロ定義を含んでいる。また、アプリケーションプログラムの終了前に終了を通知するシグナルを送ることにより重要なデータの退避や再開時に必要なデータの保存を実行することが可能になる場合があるので、その終了処理の実装のための関数も提供している。さらに、1 つの Application Container 内で使用されるプロセス ID は Application Container の外から見えるプロセス ID とは異なっているので、Application Container 外から上記の終了を通知するシグナルを送るためにプロセス ID を知る必要がある場合を考慮し、Application Container 内外でプロセス ID を対応付ける仕組み（/proc/daware）も提供している。

3-4. D-Application Monitor

D-Application Monitor は、D-Application Manager と D-Box の間に存在し、D-Application Manager からの複数の Events の相互関係を検査し、それらの Events が全体として意味する（異常な）事象を発見（Event Correlation）し、その事象に対して適切な処置(Actions)を行なうことを目的としている。

D-Application Monitor の構成

D-Application Monitor の構成は、以下の構成ブロック図（未実装も含む）で示される。

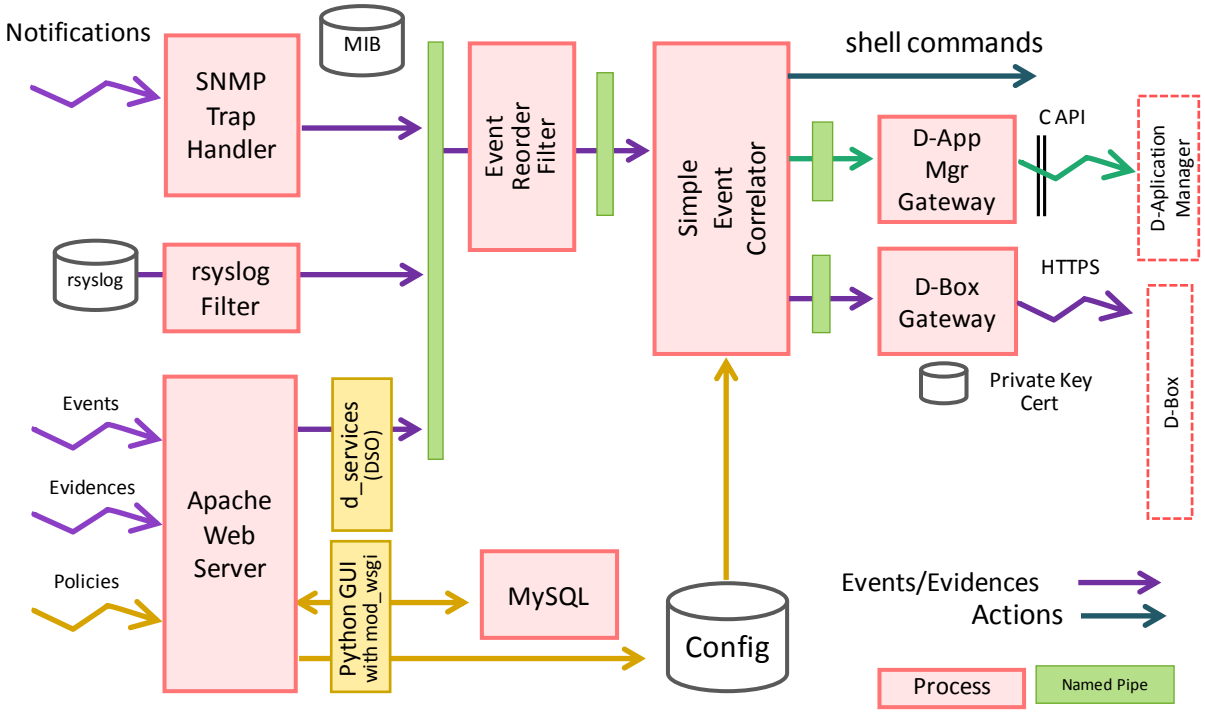

図 3. D-Application Monitor

D-Application Monitor は、Apache Web Server を経由して、D-Application Manager から Events を受け取る。将来必要があれば、SNMP の Notifications (Traps)や rsyslogd からも受け取ることができるよう拡張することができる。Apache Web Server、SNMP Notification や rsyslogd からの Events は、Named Pipe を経由して、Event Re-order Filter に送られる。Event Re-order Filter は、複数の Events ソースから遅延して届いた Events をある一定の期間内で正しい時間順に並べ直す。

時間順に並べ直された Events は、Events 間の関係を検査する(Event Correlation)を行なうために、Event Correlator に送られる。Event Correlator は、Events を受け付けて状態遷移を行なう有限状態マシン（Finite State Machine）として実装された、オープンソースの Simple Event Correlator を利用している。例えば、以下のような状態遷移とそれに伴う Actions を実装することができる。Actions としては、shell スクリプトの実行や他の Named Pipes への書き込み等が可能である。

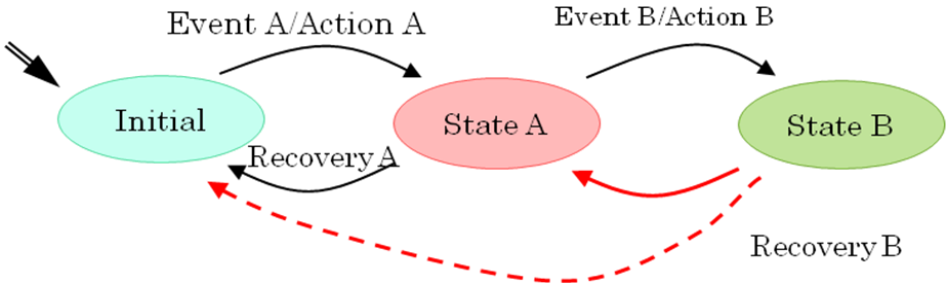

図 4. Event Correlation

Simple Event Correlator (SEC)は、http://simple-evcorr.sourceforge.net/ からダウンロードできる。

SEC の構成ファイル

SEC は、有限状態マシンと状態遷移に伴う Actions を記述した構成ファイルに基づき動作する。D-Application Monitor では、Apache Web Server 経由で、構成ファイルを Upload や編集することができる。構成ファイルの文法や例に関しては、man page (http://simple-evcorr.sourceforge.net/sec.pl.html) や FAQ（http://simple-evcorr.sourceforge.net/FAQ.html）がある。

3-5. D-Box

D-Box は、公開鍵基盤技術（PKI）を用いて、ログ情報の改竄を検出するための情報を付加して、保存する機能を提供する。D-Box は、以下のようにルート認証局（CA)が発行した公開鍵証明書を基とする階層構造の下位部分に属することにより、D-Box 自体の公開鍵の正当性を担保している。

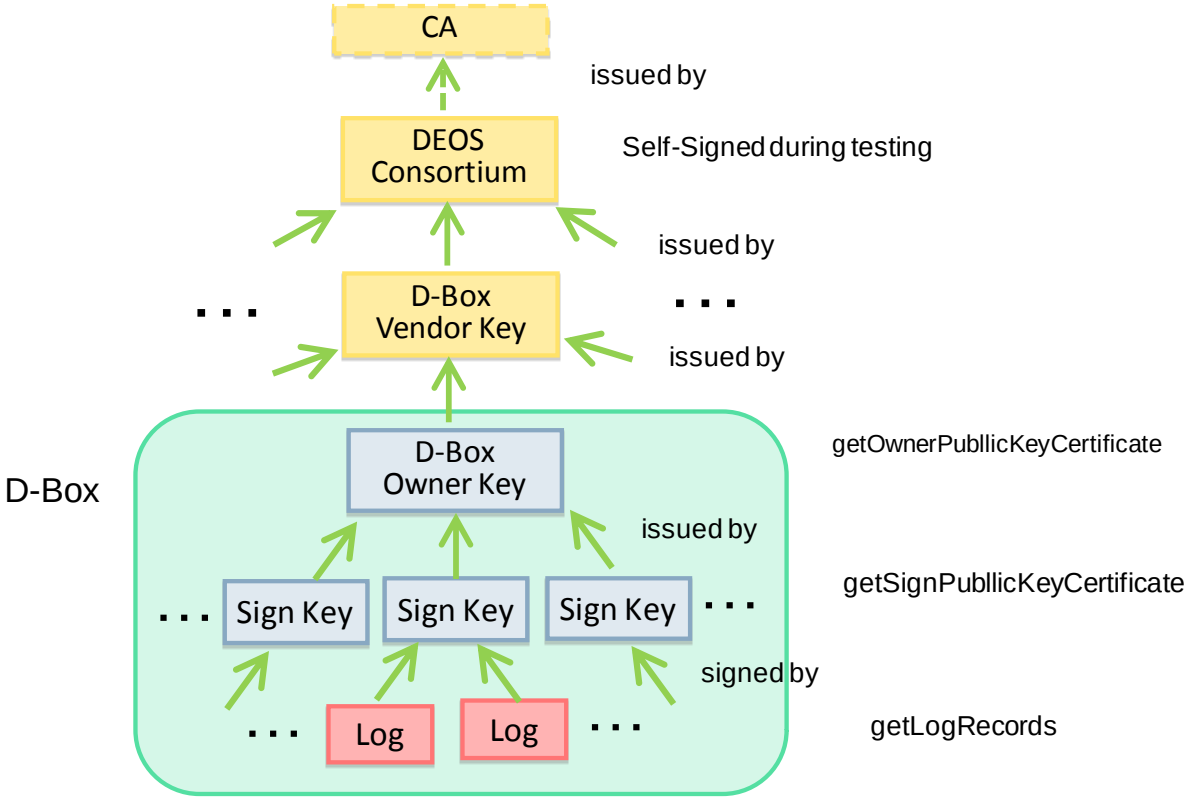

図 5. D-Box の公開鍵の正当性

D-Box の初期化時に（D-Box の製造者により）、D-Box 内に、各 D-Box に固有の D-Box Owner 鍵（PKCS #12)が導入される。D-Box の PKCS #12 は D-Box の製造者の秘密鍵で署名され、D-Box の製造者の公開鍵は、例えば、DEOS センターの秘密鍵で署名され、最終的には、信頼されたルート認証局にたどり着く。

その後、D-Box は運用時にログ情報を暗号化するための RSA 鍵の公開鍵と秘密鍵の対を生成する。この公開鍵と秘密鍵の対は、同じ秘密鍵が長期間にわたって使い続けられることを避けるために、適宜作成される。

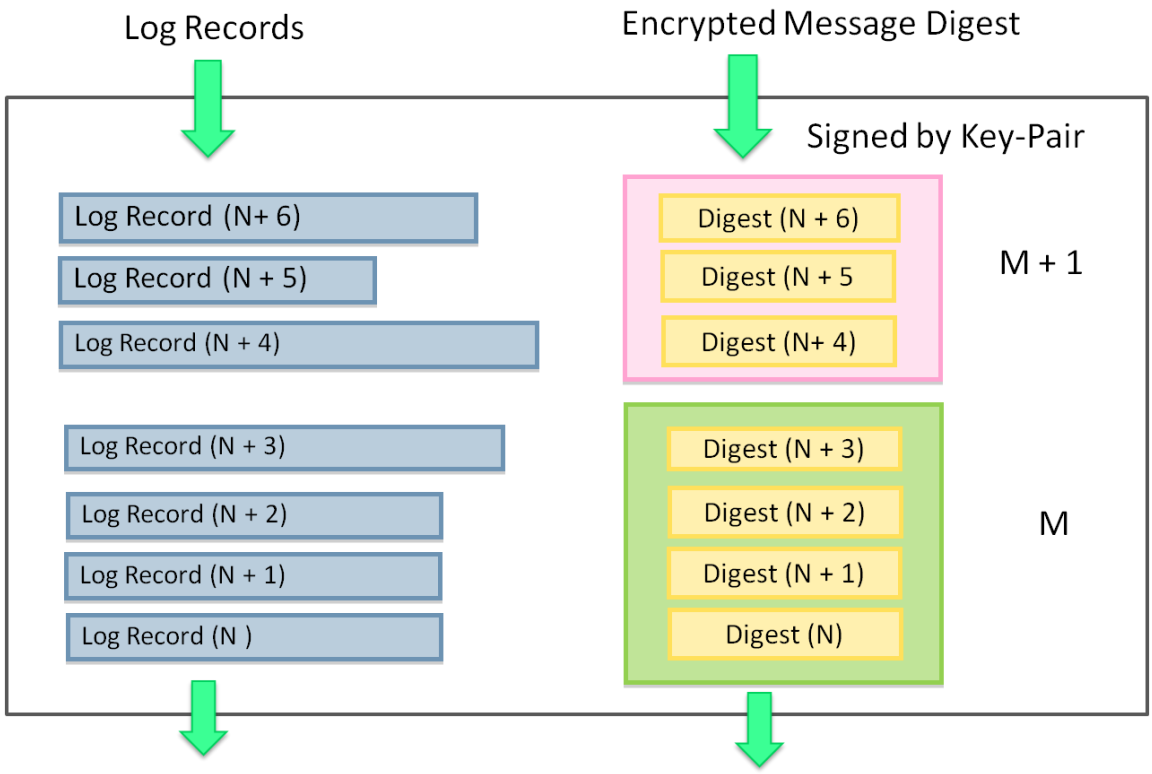

図 6. Log と Digest

D-Box は、HTTPS (TLS/SSL)経由でログ情報を受け取ると、ログ情報をバイト列（Octet String）として解釈して、MD5 によるハッシュ（Digest）を作成し、D-Box 内の最新の Sign 用秘密鍵でその Digest を暗号化する。

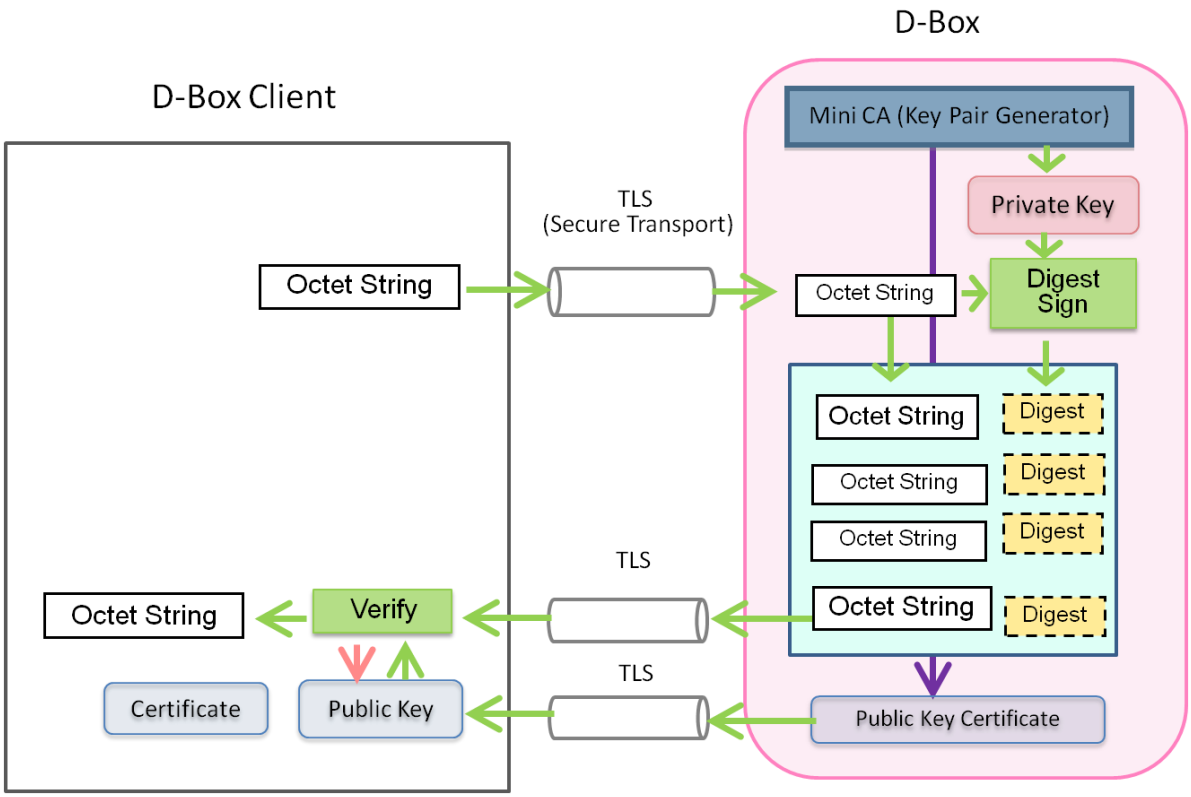

図 7. D-Box Client と D-Box

D-Box からログ情報を取り出すプログラムは、取り出したログ情報から作成した MD5 によるハッシュ（Digest）と、D-Box から受け取った暗号化されたハッシュ（Digest）を D-Box の対応する Sign 用公開鍵で復号化したハッシュを比較することにより、ログ情報の改竄を検出することができる。

D-Box の構成

D-Box は、Apache Web Server と MySQL を基に実装されている。

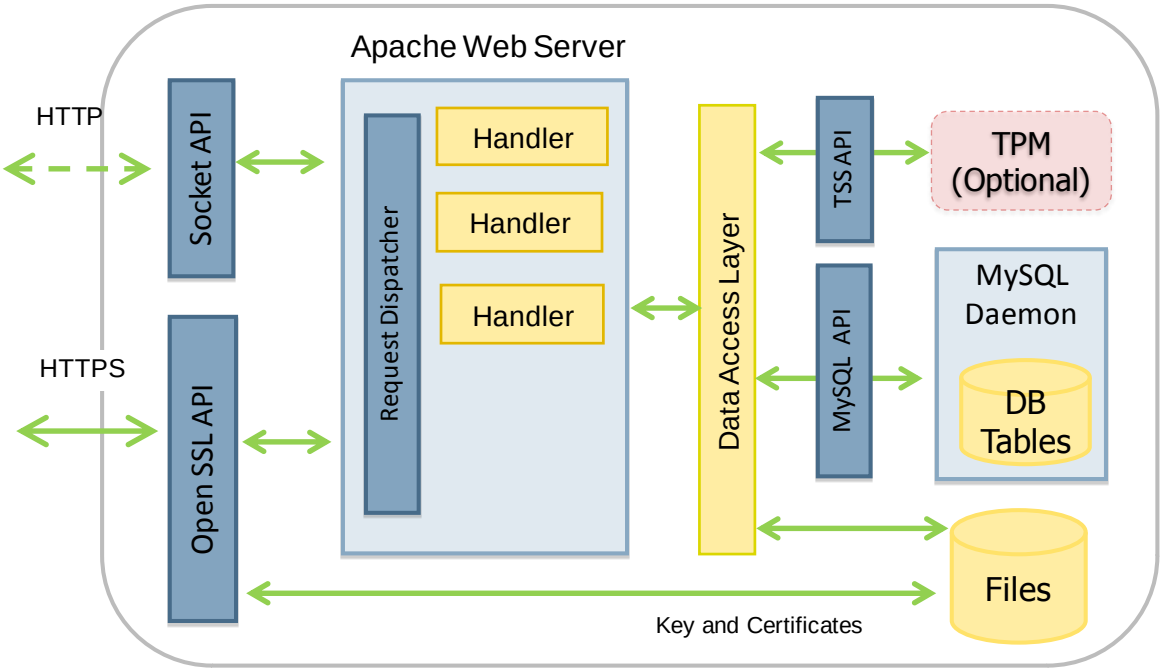

図 8. D-Box の構成

D-Box に対する操作は URL (http://d-box:8080/d_box_main.html)へ Web Browser を経由してアクセスすることにより行なうことができる。

3-6. D-Script Engine

D-Script Engine は D-Script を安全・確実に実行する役割を担っている。D-Script には、何を監視しどのように対処するか記述してあるので、それによって D-Application Manager、D-Application Monitor、D-System Monitor を制御する。

4. D-RE の使用方法

本 D-RE 仕様書では、D-Application Manager と D-Box の使用方法について以下に簡単に記述する。インストール方法については、Appendix A1.D-RE 導入ガイドを参照。

他の D-Visor、D-System Monitor、D-Application Monitor、D-Script Engine については、それぞれのドキュメントを参照。

4-1. D-Visor の使用方法

D-Visor としては D-Visor86 と Linux KVM を使用可能である。D-Visor86 はマルチコアの x86 プロセッサ上で稼働し D-System Monitor と組み合わせて使用可能である。稼働環境や導入方法等は、「D-Visor86 + D-System Monitor 環境構築手順書」を参照。

Linux KVM は、Ubuntu 上では標準でインストールされており、KVM コマンドで使用可能である。D-Application Manager のコマンド/API でも標準の Linux KVM を使用する。KVM に D-System Monitor を組み込んで使用する場合には、QEMU-KVM にパッチをあててビルドする必要がある。QEMU-KVM の修正方法や使用方法の詳細については、「QEMU-KVM + D-System Monitor 環境構築手順書」を参照。

4-2. D-System Monitor の使用方法

D-System Monitor を使用するには、D-Visor86 あるいは修正した QEMU-KVM を VMM として使用する必要がある。それぞれの使用方法は、「D-Visor86 + D-System Monitor 環境構築手順書」あるいは「QEMU-KVM + D-System Monitor 環境構築手順書」を参照。

4-3. D-Application Manager の使用方法

D-Application Manager は主にコンテナ機能を提供し、以下のコマンドとライブラリから構成される。

4-3-1. コマンド

dre-core パッケージをインストールすると、以下のコマンドが/usr/sbin ディレクトリの下に導入される。

- dre-sys コマンド
  System Container について、以下の機能を利用できる。
  ・コンテナの生成・開始・停止・破棄
  ・checkpoint/restart 機能、snapshot の save/load 機能
  ・migration 機能により System Container を他マシン上に移動
  ・コンテナの一覧の取得
  ・コンテナの状態（開始されているかどうか等）の取得
  ・コンテナのリソース情報の取得・制限

- dre-app コマンド
  Application Container について、以下の機能を利用できる。
  ・コンテナの生成・開始・停止・破棄
  ・snapshot の save/load 機能
  ・コンテナの一覧の取得
  ・コンテナの状態（開始されているかどうか等）の取得
  ・コンテナのリソース情報の取得・制限
  ・cgroup の作成・削除
  ・cgroup リソース情報の取得・制限

  コマンド引数等の詳細は、インストール後に /usr/share/doc/dre-core ディレクトリの下に置かれるドキュメントや D-RE コマンド仕様書（DEOS-FY2013-EC-01J）を参照。

4-3-2. ライブラリ

libdre0 パッケージをインストールすると、以下の 2 つの C 言語用ライブラリ libdresys と libdreapp が/usr/lib ディレクトリの下に導入される。また、libdre0-dev パッケージをインストールすると、ライブラリを使用したプログラムを作成するためのヘッダファイルが/usr/include ディレクトリの下に導入される。

- libdresys ライブラリ
  System Container について、以下の機能を利用するための関数を提供する。
  ・コンテナの生成・開始・停止・破棄
  ・checkpoint/restart 機能、snapshot の save/load 機能
  ・migration 機能により System Container を他マシン上に移動
  ・コンテナの一覧の取得
  ・コンテナの状態（開始されているかどうか等）の取得
  ・コンテナのリソース情報の取得・制限

- libdreapp ライブラリ
  Application Container について、以下の機能を利用するための関数を提供する。
  ・コンテナの生成・開始・停止・破棄
  ・snapshot の save/load 機能
  ・コンテナの一覧の取得
  ・コンテナの状態（開始されているかどうか等）の取得
  ・コンテナのリソース情報の取得・制限
  ・cgroup の作成・削除
  ・cgroup リソース情報の取得・制限

  関数名や関数の引数・戻り値等の詳細は、インストール後に /usr/share/doc/libdre0-dev ディレクトリの下に置かれるドキュメントや D-RE API 仕様書（DEOS-FY2013-EA-01J）、D-RE API サンプルプログラム解説書（DEOS-FY2013-SP-01J）を参照。

libdaware0 パッケージをインストールすると、C 言語用ライブラリ libdaware が/usr/lib ディレクトリの下に導入される。また、libdaware0-dev パッケージをインストールすると、ライブラリを使用したプログラムを作成するためのヘッダファイルが/usr/include ディレクトリの下に導入される。

- libdaware ライブラリ
  application program を開発する際に以下の機能を利用するための関数を提供する。
  ・シグナルに対するコールバック関数を指定する
  ・ログ出力を容易にする

  関数名や関数の引数・戻り値等の詳細は、インストール後に /usr/share/doc/libdaware0-dev ディレクトリの下に置かれるドキュメントや D-RE API 仕様書（DEOS-FY2013-EA-01J）、D-RE API サンプルプログラム解説書（DEOS-FY2013-SP-01J）を参照。

daware-mod パッケージをインストールすると、daware カーネルモジュールが導入され、Application Container の内外でプロセス ID を対応づけるための特殊デバイス/dev/daware が使用可能になる。libdaware ライブラリを使用してシグナルに対するコールバック関数を指定する際に、Application Container 外からシグナルを送るためにプロセス ID を知る必要がある場合に使用する機能である。
詳細は /usr/share/doc/daware-mod ディレクトリの下に置かれるドキュメントを参照。

4-4. D-Application Monitor の使用方法

D-Application Monitor は、対象となるシステムに合わせて実装する必要がある。DEOS プロジェクトでは CD ショッピングサイトに適用したサンプル実装を開発し提供している。その詳細については、DEOS Programming Reference (DEOS-FY2013-PR-01J)を参照。

4-5. D-Box の使用方法

DEOS プロジェクトでは D-Box のサンプル実装を開発中で、2013 年 9 月に提供予定。

4-6. D-Script Engine の使用方法

D-Script Engine はスクリプトを安全・確実に実行するスクリプト言語エンジンである。使用方法の詳細については、スクリプト言語エンジンのドキュメントを参照。

5. おわりに

D-RE はステークホルダ合意に基づくオープンシステムディペンダビリティ（OSD）を実現するサービスを提供するための実行環境である。本 D-RE 仕様書では、第 1 章では、OSD の実現のために求められる機能について一般的に記述し、第 2 章では、求められるシステム動作と D-RE の機能の対応、第 3 章では、D-RE 各モジュールの機能、第 4 章では、公開されている D-RE 各モジュールの使用方法について紹介した。

D-RE はまだ研究開発中であり、現在公開されているものは、1 つのサンプル実装に過ぎない。そのため、提供されるソフトウェアは、「AS IS (無保証)」で提供されているものであり、明示、暗黙を問わず、いかなる保証も提供しない。

Appendix

A1. D-RE 導入ガイド

A1-1. 動作環境

動作環境は、以下の通りです。

OS:	Ubuntu 12.04 LTS (i386/amd64)
CPU:	Intel-VT または AMD-V 対応の CPU
RAM:	2GB 以上(システムコンテナ 1 個につき、最低 512MB 以上必須)
Network:	DHCP による IP アドレス取得が出来ること

A1-2. パッケージの入手

Ubuntu 12.04 LTS 用の実行モジュールとソースコードを、www.dependable-os.net から取得可能です。

A1-3. インストール手順

パッケージのインストール手順は、以下の通りです。

A1-3-1. apt の設定

1. /etc/apt/sources.list に、以下の 2 行を追加します。

```
deb http://www.dependable-os.net/tech/D-RE/package precise universe
deb-src http://www.dependable-os.net/tech/D-RE/package precise universe
```

2. リポジトリを更新します。

```
$ sudo apt-get update
```

A1-3-2. パッケージのインストール

以下のコマンドで、必要なパッケージをインストールします。

1. 必要最低限のパッケージ群

```
$ sudo apt-get install dre
```

2. 開発者用パッケージ(オプション)

```
$ sudo apt-get install libdre0-dev libdaware0-dev
```

A.1-3-3. OS の再起動

OS を再起動します。

```
$ sudo reboot
```

A1-4. 初期設定

A.1-4-1. 設定ファイルの編集

/etc/dre.conf を適宜編集します。(通常は不要。)

DRE_BRIDGE_IF: ブリッジインタフェース名
DRE_DEFAULT_SSHD_PORT: アプリケーションコンテナが使用する SSHD ポートのデフォルト値
DRE_SYS_DEFAULT_BASE: システムコンテナがデフォルトで使用するベースイメージ
DRE_SYS_DEFAULT_NETMASK:システムコンテナのネットマスクのデフォルト値
DRE_SYS_DEFAULT_GATEWAY:システムコンテナが使用するゲートウェイのデフォルト値
DRE_SYS_DEFAULT_DNS: システムコンテナが使用する DNS サーバのデフォルト値
DRE_SYS_DEFAULT_MAC: システムコンテナに割り当てる MAC アドレスのデフォルト値
DRE_SYS_DEFAULT_MEMORY: システムコンテナに割り当てるメモリ容量のデフォルト値

A.1-4-2. システムコンテナのベースイメージの作成

システムコンテナがベースイメージとして使用する、KVM イメージを作成します。

1. 以下のコマンドを実行します。コマンドの実行には、数十分掛かる場合があります。

```
$ sudo dre-base
```

A.1-4-3. ブリッジの設定

/etc/dre.conf で設定したブリッジインタフェース(デフォルトは br0)に、任意のネットワークインタフェースをブリッジします。

1. /etc/network/interfaces を編集します。(下記は、eth0 を br0 にブリッジする場合の設定例)

```
auto lo
iface lo inet loopback

auto br0
iface br0 inet dhcp
bridge_ports eth0
bridge_stp off
bridge_maxwait 1

auto eth0
iface eth0 inet manual
up ifconfig eth0 0.0.0.0 up
```

2. network を再起動します

```
$ sudo /etc/init.d/networking restart
```

A.1-4-4. daware.mod カーネルモジュールのインストール

daware.mod をインストールします。この作業は、カーネル更新時に毎回実行する必要があります。

2. 以下のコマンドを実行します。

```
$ sudo /var/lib/dre/daware-mod/install.sh
```

以上で D-RE のインストールと設定が完了です。

D-RE の使用にあたっては、DEOS ホームページ(http://www.dependable-os.net/) で公開している以下のドキュメントを参照してください。

- D-RE API 仕様書 （文書番号 DEOS-FY2013-EA-01J）
- D-RE コマンド仕様書 （文書番号 DEOS-FY2013-EC-01J）
- D-RE API サンプルプログラム解説書 （文書番号 DEOS-FY2013-SP-01J）


Dependable Embedded Operating Systems
DEOS
Dependability Engineering for Open Systems

DEOS プロジェクト