DEOS-FY2013-RE-01E

D-RE (DEOS Runtime Environment) Specification

< Appendix > D-RE Installation Guide

Version E1.0

2013/07/15

Edited by DEOS R&D Center

DEOS Project

JST-CREST

Research Area

"Dependable Operating Systems for Embedded Systems Aiming at Practical Applications"

Japan Science and Technology Agency

(Blank Page)

Contents

1. D-RE Overview

In order to realize Open Systems Dependability (OSD), in other words, in order to continuously maintain the services of ever-changing systems and to maintain accountability for the system operations and processes, the following is required:

- An iterative process which connects development and operation for continuous improvement (DEOS Process)
- A consensus building support tool (D-Case) and a runtime environment (D-RE: DEOS Runtime Environment)
- Scripts (D-Script) and application programs (D-Aware Application Program) which realize OSD based on D-Case

This D-RE Specification describes the second item above, especially the DEOS Runtime Environment (D-RE). Chapter 1 describes functions to realize OSD in general, and functions that have been developed and their implementation are described in the following chapters.

DEOS Runtime Environment (D-RE) is a runtime environment which provides services to realize dependability based on stakeholder's agreements. It is shown in the lower right in the DEOS architecture shown below (Fig. 1). As shown in the diagram, the D-RE consists of various subsystems.

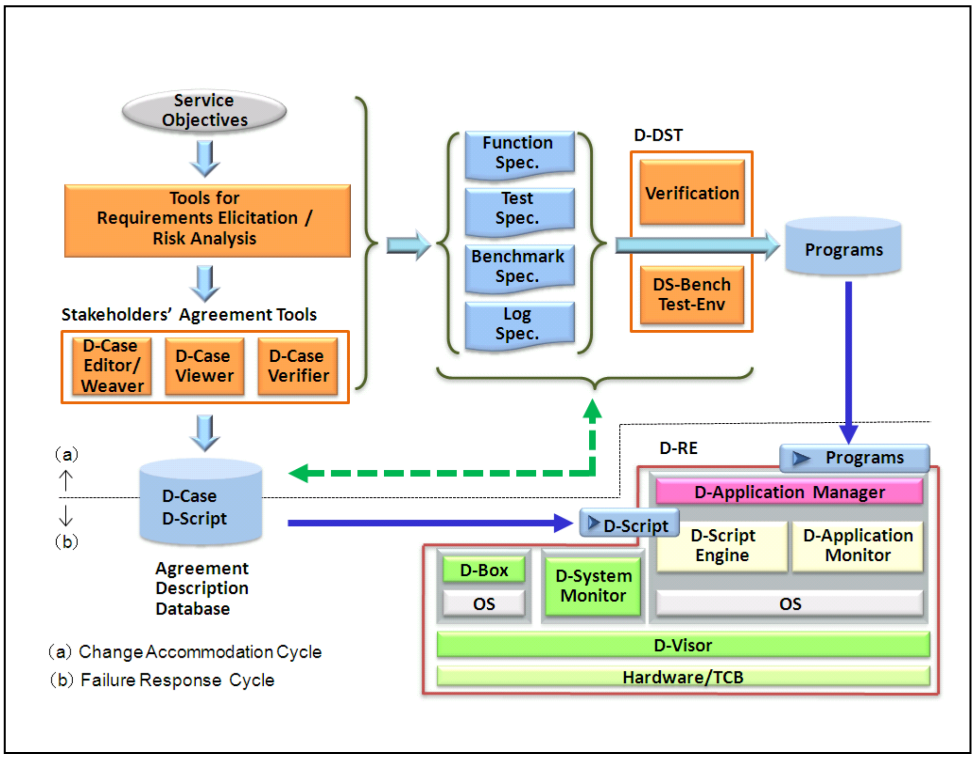

Fig. 1 DEOS Architecture

D-Visor provides a mechanism (System Container) to isolate components in a system. A system container is a Virtual Machine (VM) with an OS running in it. Errors or failures of a system container don't affect the other system containers. In one case, a D-Visor might run directly on hardware. But in the case where Linux KVM is used to provide system containers, an OS runs on hardware and a D-Visor runs on the OS.

D-System Monitor provides functions to monitor a system container. It watches a VM from the outside and detects abnormal behavior of the OS which may be caused by malware, without affecting the OS running in the system container.

D-Application Manager provides container mechanisms, and so can isolate application programs in a system. There are two kinds of containers: system containers and application containers. The system container is a VM provided by D-Visor, but the application container is a lightweight container, not a VM. Each container may have limitations on resources, such as CPU usage and memory usage.

D-Application Monitor watches the states of applications, collects evidence, and passes them to the D-Box.
D-Box stores the evidence and other data essential to achieve OSD, in a safe and secure way.
D-Script Engine executes D-Script safely and controls D-Application Manager, D-Application Monitor, and D-System Monitor, according to D-Script prescriptions about what to monitor and how to handle the monitoring results.

2. Required System Functions

In order to realize the normal operations and failure responses comprising dependable systems, various system functions are required. D-RE provides basic functions to implement the following requirements of dependable systems.

2-1. Application Monitoring

Using D-Script, system providers describe what to monitor/detect, how to act upon failure predictions, and failure detection suitable for each application program. Although the data to be monitored/detected varies depending on the application program, D-RE is prepared with monitoring functions for CPU usage and memory usage of a container (system container and application container) , as general functions. Therefore, CPU usage and memory usage of an application program can be monitored indirectly when the application program is running in a container.

2-2. System Monitoring

A system container provided by D-RE is a Virtual Machine (VM). CPU usage monitoring and memory usage monitoring functions may be used to verify that the VM and its guest OS are running normally. But commands or tools for system monitoring in the guest OS must be used to verify whether or not services in the guest OS are actually operating normally.
Without affecting the guest OS running in the VM, the D-System Monitor in D-RE watches a VM from the outside and detects abnormal behavior of the guest OS caused by malware. D-System Monitor can monitor I/O data in virtual devices and the memory contents of the guest OS, using functions embedded in the Hypervisor (such as D-Visor86 and Linux KVM described in the chapter 3).

2-3. Evidence Logging

D-RE provides a component, called D-Box, into which data may be stored by particular programs which are authenticated and authorized to have read/write access to the D-Box. The D-Box hashes the data passed to the D-Box with MD5 algorithm, encrypts the hash value with an RSA private key of the D-Box, and stores both the data and the encrypted hash value. When another program retrieves the data from the D-Box, both the data itself and the encrypted hash value are returned to the program, and the program can verify the integrity of the data with the encrypted hash value and the RSA public key of the D-Box corresponding to the private key used to encrypt the hash value. Because the storage capacity of the D-Box is limited, the data which are to be stored in the D-Box must be carefully selected so that the storage is not exhausted.
Regardless of whether or not the log records are stored in a D-Box, log records from application programs are useful in locating and identifying the root causes of failures and faults. To make

consolidating and correlating log records from various application programs easier and more efficient, the format of the log records is unified. The unified format will also make possible increased use of pattern matching techniques to analyze the log records automatically. D-RE provides a set of functions to write such log records.

2-4. Isolation

When developing a system, we need to consider system structure in which a failure of a part of the system doesn't affect all of the system and in which a failed part in the system can be easily isolated from the other parts. Failure response action in Open Systems Dependability needs to isolate a failed part from the rest of the system and to continue the other services of the system uninfluenced by the failure.

With D-RE, an OS in a system container and process groups in an application container can be started/stopped. System containers are Virtual Machines (VM), and an independent guest OS is running in each VM. A failure of a guest OS in a system container doesn't affect the other guest OSs. The application container is a lightweight container based on Linux Container (LXC). Process groups in an application container and process groups in the other application containers don't affect each other.

2-5. Quota

Using a container of D-RE, CPU usage and memory usage of a program in the container can be limited without affecting the other containers. This functionality prevents the situation that a program in a container goes out of control and exhausts CPU resources so that the other programs cannot get enough CPU resources, or the situation where a container exhausts large memory area and the other containers cannot be allocated enough memory area. The resource quota of D-RE can be changed dynamically after starting a container. Therefore, according to monitoring results, resource quota can be used to continue services of a system without failures, though services might be slower than usual.

2-6. Undo

Systems are sometimes updated to introduce new services or to fix bugs. But, even if enough tests are done for the updated code, sometimes the code doesn't work as expected. In the worst case, the system may go down and cannot continue to provide services to users. If the state of the system before applying updates is recovered, the same services as before can be provided to users. For such situations, undo functionality is important.

It is not generally easy to cancel a sequence of operations in reverse order. Therefore, the undo is sometime implemented by storing the environment status data before the sequence of the operations and restoring that environment status data. D-RE supports this type of undo function, which stores a status of a system container or an application container at a certain point of time and restarts the container after restoring the stored status.

There are two operations needed to store an environment status of a container. One is done after completely stopping the container, and the other is done when suspending the container temporarily. When the container is completely stopped, all programs in the container are also stopped and the resumption of a prior state is same as the normal start of the container. On the other hand, when the container is suspended temporarily, only CPU allocation is temporarily stopped and all programs in the container are still running and keep all their contents in the memory, and therefore after the resumption, the execution of the programs using the preserved memory contents continues. In D-RE, the former is called the save/load function of a snapshot, and the latter is called the checkpoint/restart function.

2-7. Migration

Using D-RE, a guest OS in a system container can be moved to another container by the system container. This is known as migration of a VM. The migration can be done between computers connected over a network, as well as within one computer. In order to make a system container migrate to another computer connected over a network, another system container must be available in the target computer. Live migration, which moves a running guest OS, is supported, as well as general migration, which moves a stopped guest OS. The live migration is usually used to continue services during hardware maintenance, not to avoid failures caused by hardware troubles.

3. Design Concept and Structure of D-RE Subsystems

D-RE provides basic functions to construct dependable open systems, and consists of the following subsystems.

3-1. D-Visor

D-Visor is a virtual machine monitor and provides a mechanism (system container) to isolate components in a system. A system container is a Virtual Machine (VM) and a guest OS is running in it independently. Errors or failures in a system container don't affect the other system containers.

D-Visor86 developed by the University of Tsukuba is a virtual machine monitor for multi-core processors. It can execute a modified Linux in each processor core of a multi-core (including hyper-threading enabled) x86 CPU. D-System Monitor runs on Linux in one of processor cores and monitors guest OSs (Linux) in the other processor cores.

ART-Linux developed by the National Institute of Advanced Industrial Science and Technology (AIST) divides computational resources and allocates them statically to OSs during initialization. It executes an SMP-Linux and multiple ART-Linuxes in physical CPU cores.

APIs and commands of D-Application Manager use the Linux KVM in D-Visor. An independent OS is running in each VM, and start/stop of a system container doesn't affect the other system containers. Moreover, different guest OSs can run in each system container. It may be possible to limit failure impact if independent unrelated applications run on separated system containers.

3-2. D-System Monitor

D-System Monitor provides monitoring functions in cooperation with D-Visor. It monitors a VM from the outside and detects abnormal behavior of a guest OS which may be caused by malware, without affecting the behavior of the guest OS.

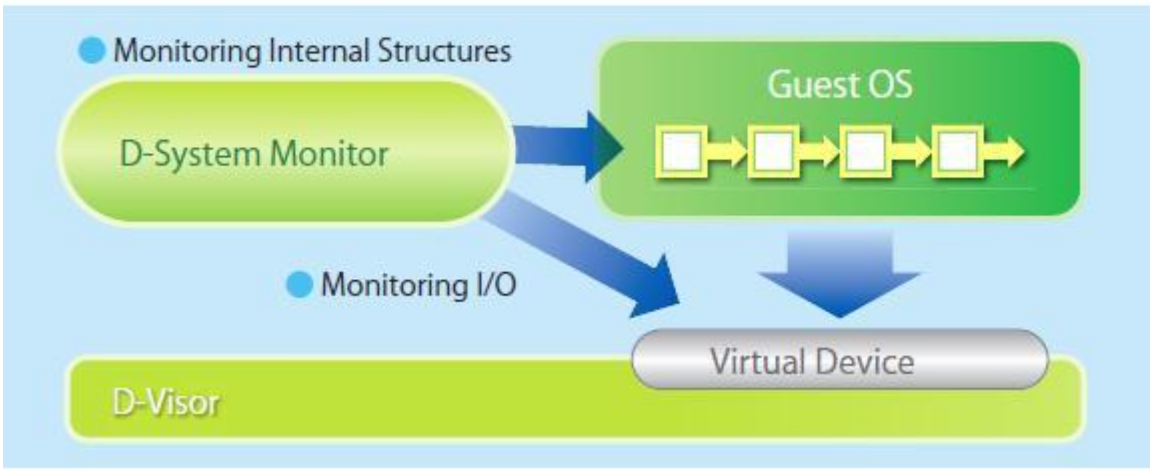

Fig. 2 D-System Monitor

D-Visor provides D-System Monitor with functions to monitor I/O requests of VM and to directly access the memory of the guest OS. D-System Monitor uses data acquired through these functions to detect abnormal behavior of the guest OS. At this time, the following three monitoring mechanisms are available.

- FoxyKBD: Monitors OS behavior during key inputs. Evaluating the relation between I/O requests and large amounts of simulated key inputs, it detects malware which intercepts key inputs.
- RootkitLibra: Detects hidden files. It compares file meta data acquired in the guest OS with the data in NFS packets for files under NFS-mounted directories and detects hidden files and incorrect file size.
- Waseda LMS: Detects hidden processes. It directly accesses the memory of the guest OS, compares data in the task list with data in the run queue in the guest kernel, and detects hidden processes.

These three monitoring mechanisms are available in the D-System Monitor executed with the D-Visor86 developed by the University of Tsukuba. Furthermore, DEOS Project developed a modified QEMU-KVM to embed the D-System Monitor and to run the three monitoring mechanisms.

3-3. D-Application Manager

D-Application Manager provides containers as a mechanism to isolate applications. There are two types of containers, system container and application container. D-Application Manager also provides functions in libraries to support log output and termination processing of application programs.

System containers are Virtual Machines (VM) provided by D-Visor. While D-Visor86 and Linux KVM can be used as D-Visor as described before, D-Application Manager uses VMs of Linux KVM as system containers controlled through APIs and commands. When Linux KVM is used as D-Visor, an OS (Linux) runs on hardware and D-Visor runs on the OS. Therefore, control functions of the D-Visor or commands in D-Application Manager running on the same OS can be provided as APIs.

An independent guest OS runs in each system container, and a failure of a guest OS in a system container doesn't affect other guest OSs. The resource quota of CPU usage and memory usage can be set for each system container, so that appropriate resource allocation of the system to prevent exhaustion of resources is achieved.

A system container has both the checkpoint/restart function and the save/load function of snapshot. Checkpoint/restart function saves data while suspending the system temporarily. Only CPU allocation is temporarily stopped and the guest OS and all processes in the container are still running and keep all their contents in the memory, and therefore the resumption continues the execution of the guest OS and the processes using the preserved memory contents. On the other hand, because save/load function of snapshot stores data of the file system after the guest OS in the VM stops, the resumption is same as a boot of the OS. When the system status before introducing an update program or data is stored using the checkpoint/restart function or save/load function of snapshot, the system can be restored to the former status in case of a failure.

Furthermore, system containers have migration function, which moves a guest OS in a system container to other system container. It is known as migration function of VM. The migration can be done between system containers in computers connected through a network, as well as within one computer. Live migration, which moves a running guest OS, is supported, as well as general migration, which moves a stopped guest OS.

When creating a system container, a KVM image is used. Its base image can be created with D-RE commands. Even without using D-RE, however, a KVM image prepared by a user can be used to create a system container.

An application container is a lightweight container, not a VM. D-Application Manager uses LXC containers as application containers controlled through APIs and commands. Because all containers run on an OS, a failure of the OS affects all containers. However, processes in one container are not affected by processes of the other containers. Using an application container, some processes can be handled as a group and each group can have limitations on CPU usage and memory usage. Even if it is convenient for some application programs to run together on an OS, when each of the processes runs in different application containers, the segments of resources such as CPU and memory used by each process can be separated from each other and bad effects of one process failure upon the other processes can be reduced.

Application containers have the save/load functions of snapshot. But checkpoint/restart functions of application containers are not supported by D-Application Manager, because checkpoint/restart functions of LXC have not been implemented yet.

Both system containers and application containers are used in sequence: create, start, stop, destroy. Operations upon a container can be specified through commands, such as **dre-sys** and **dre-app**, and libraries, such as **libdresys** and **libdreapp**. In addition, both checkpoint/restart functions and save/load functions of snapshot can be used through commands and libraries.

D-Application Manager also provides the **libdaware** library to support log output and termination processing of application programs. It includes some functions and macros to set the log output format to be syslog. It also includes functions to process termination of application programs, so that important data can be stored when a termination signal is sent before actual process termination. Moreover, though a process ID used in an application container is different from a process ID seen outside of the container, sometimes one of the signals above needs to be sent from outside of the container. For this situation, D-Application Manager provides a mechanism (/proc/daware) to associate a process ID in a container with one outside.

3-4. D-Application Monitor

A D-Application Monitor, which is located between D-Application Managers and a D-Box, receives events from the D-Application Managers and handles them collectively to decide upon and take the action which is the most appropriate response to the received events. The processing of those events by the D-Application Monitor includes so-called Event Correlations and Event Aggregations.

Implementation of D-Application Monitor

The following diagram shows one implementation of D-Application Monitor, and also those components which are not yet implemented:

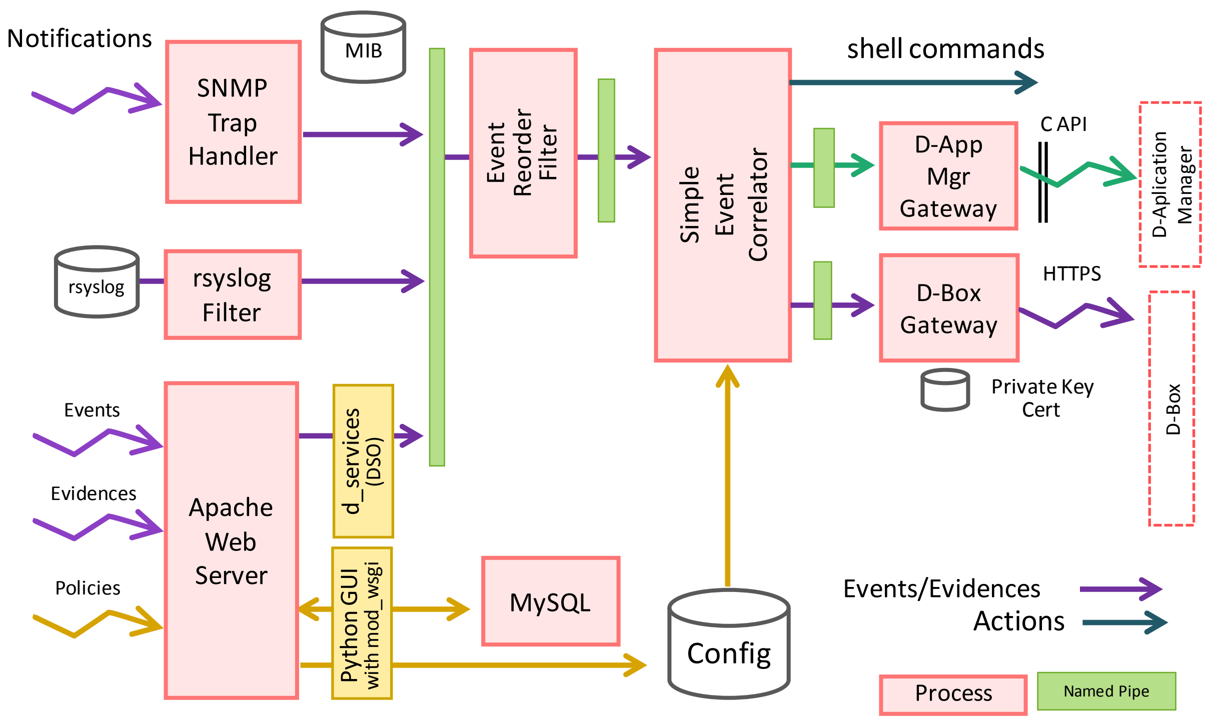

Fig. 3 D-Application Monitor

The D-Application Monitor receives events from D-Application Managers through an Apache Web Server. In the future, it may be enhanced to accept notifications (traps) from SNMP agents and system log daemon (rsyslogd). Events from different event sources may take different time periods to reach the D-Application Monitor because of delays in passing through various components. Hence, an Event Re-order Filter will reorder events from the various event sources according to a timestamp in each event within a certain time-window, which will be specified to balance the requirements to react to events as soon as possible and to allow for typical delays in the paths.

The re-ordered events are then processed by an Event Correlator, which handles sequences of events using (finite) state machines in order to guess feasible causes and decide the most appropriate action to deal with the events. The Event Correlator is implemented with the Simple Event Correlator (SEC), which is available at http://simple-evcorr.sourceforge.net/. The following diagram shows a simple finite state machine for handing a sequence of events consisting of an Event A followed by an Event B:

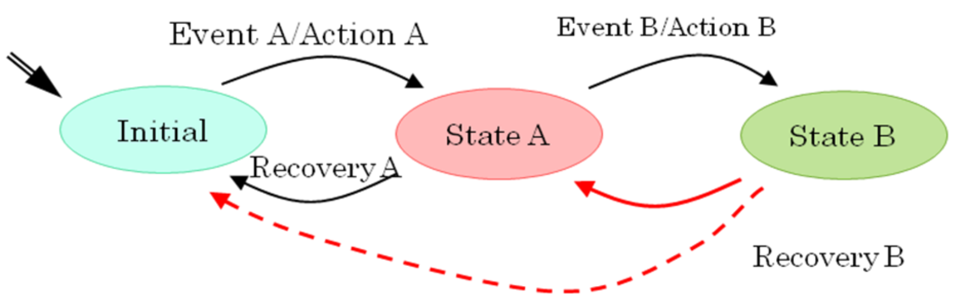

Fig. 4 Event Correlation

When an Event A occurs in the Initial state, the machine takes Action A and goes to State A, for which a timeout value is specified. In State A, if an Event B occurs within the timeout value, the machine takes Action B and goes to State B. However, if no Event B occurs within the timeout value, the machine takes Recovery A action and returns to the Initial state.

Configuration Files for SEC

The SEC is designed to simulate multiple (finite) state machines according to configuration files, which define machine states and actions associated with state transitions and/or states. See the SEC manual page (http://simple-evcorr.sourceforge.net/sec.pl.html) and SEC FAQ: http://simple-evcorr.sourceforge.net/FAQ.html

3-5. D-Box

A D-Box uses several RSA private keys to encrypt hash (digest) values of data stored in the D-Box so that other parties using the D-Box can verify the integrity of the data retrieved from the D-Box by recomputing the hash value and verifying it with the public key of the D-Box. A D-Box uses Public Key Infrastructure (PKI) for other parties using the D-Box to verify the validity of a public key of the D-Box. For instance, the following series of RSA key pairs may be used:

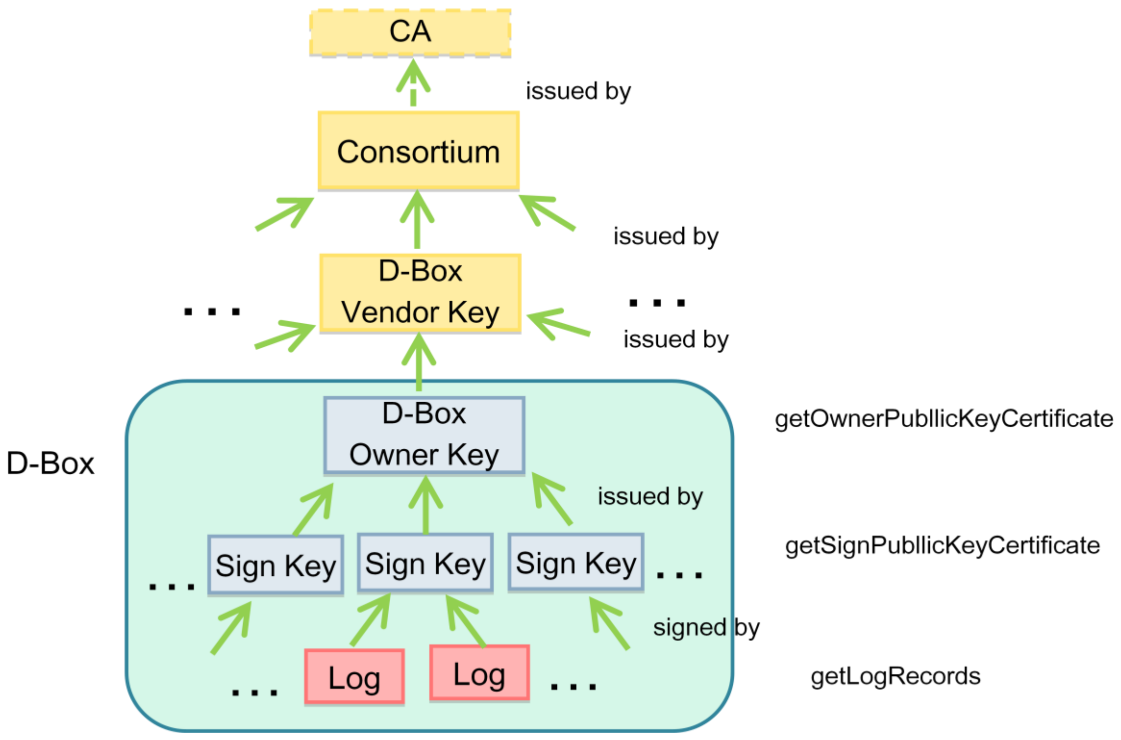

Fig. 5 Validation of a Public Key of a D-Box

A RSA private key may be installed into a D-Box as follows: A manufacturer of D-Boxes creates a pair consisting of an RSA private key and a public key for each D-Box. It will create a public key certificate for the public key which is signed with the public key certificate by the manufacturer itself, and which is obtained from a certificate authority (CA) or trusted third party (TTP), such as VeriSign. The manufacturer then creates a PKCS #12 file from the RSA private key and the corresponding public key certificate, and installs it in the D-Box.

After the initial PKCS #12 file is installed, the D-Box generates RSA key pairs for signing log records. New RSA key pairs will be generated to avoid using the same key pair repeatedly.

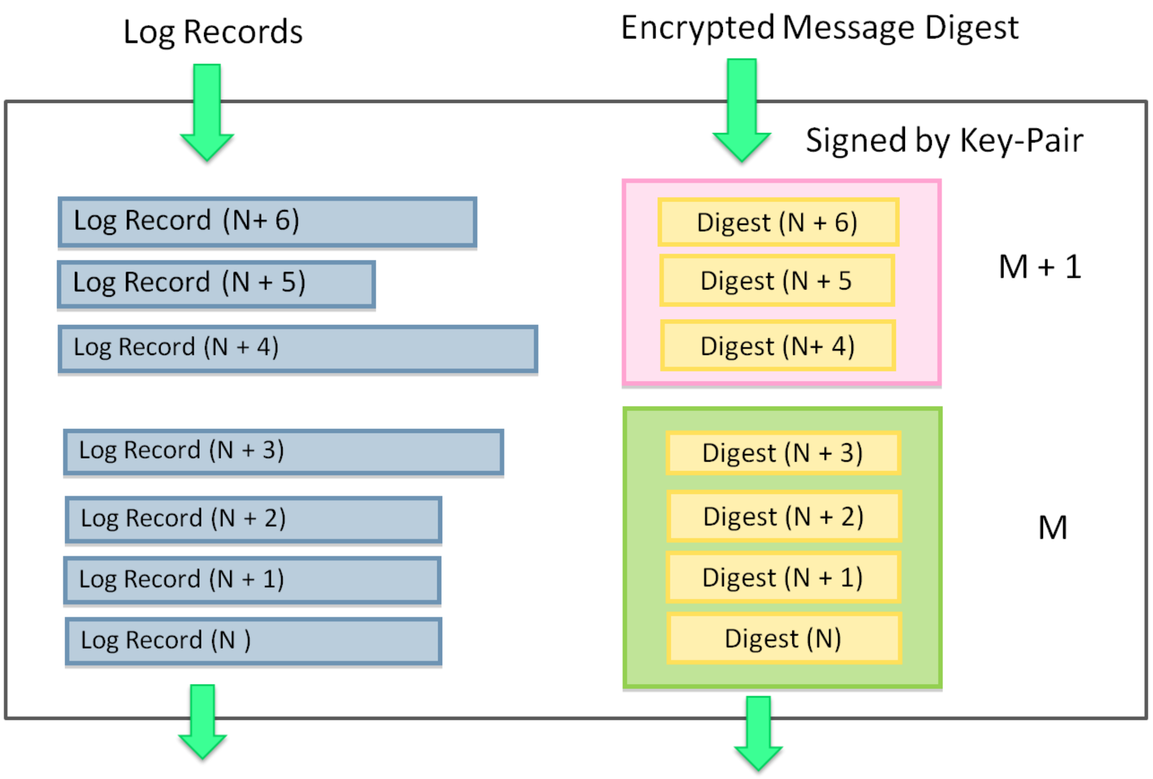

Fig. 6 Log Records and their Digests

A D-Box uses the HTTPS (TLS/SSL) protocol to receive log records and to authenticate the sender of the log records. After receiving a log record, the D-Box generates a hash (digest) value of the content of the log record interpreted as an octet string with MD5, encrypts the hash value with the latest private key among key pairs for signing of the D-Box, and stores the log record with the encrypted hash value.

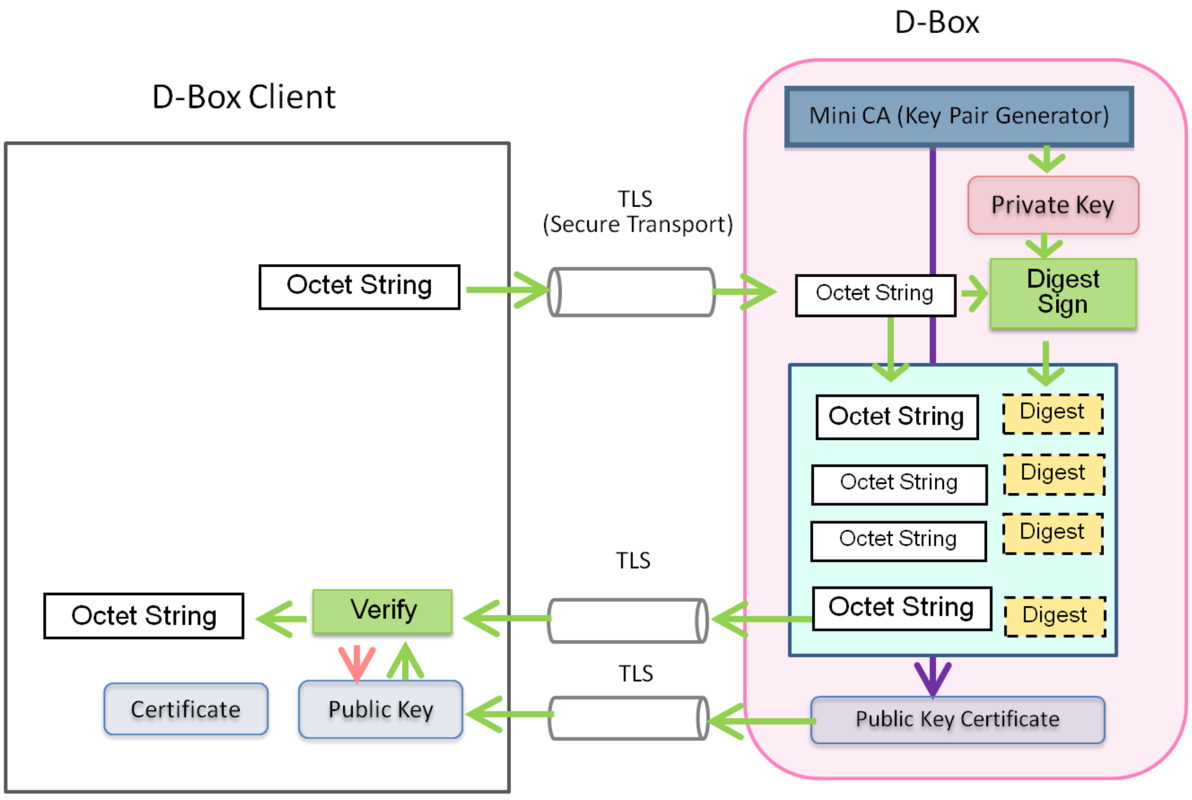

Fig. 7 Verification of Log Records by a D-Box Client

A program which accesses log records from a D-Box retrieves both the log records and the encrypted hash values corresponding to the log records, and verifies that the log records are not tampered by computing the hash value of the retrieved log record and comparing that value with the value of the retrieved hash value decrypted with the public key corresponding to the private key used by the D-Box to encrypt the hash value.

The following XML elements are an example of a Log Record, in which <d_box:key_serial> element indicates the RSA key pair used to encrypt the MD5-derived hash (digest) value of XML elements which have "digest" attributes:

```
<d_box:log_record>
  <d_box:recorded_time digest="1">2013-05-29T02:05:20.933426+00:00</d_box:recorded_time>
  <d_box:log_serial digest="2">47</d_box:log_serial>
  <d_box:key_serial digest="3">3</d_box:key_serial>
  <d_box:information digest="4">This is an example of a log record.</d_box:information>
  <d_box:digest algorithm="md5">
       lAYgRDnUCTVi    … (Omitted) … nv4ilUg7Leh6dvc1gh=
  </d_box:digest>
</d_box:log_record>
```

Implementations of D-Box

D-Box may be implemented in various ways depending on performance, storage capacity, hardware cost and security requirements. The following shows one example:

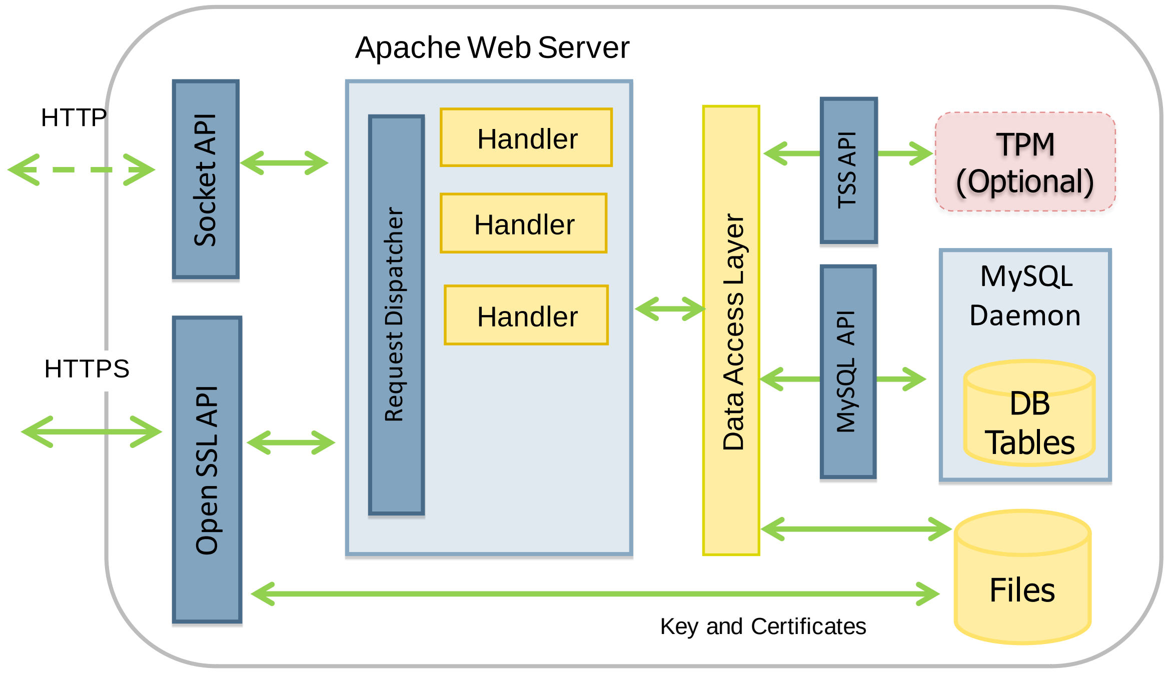

Fig. 8 An implementation of a D-Box

A prototype of a D-Box which is written in Python and Google AppEngine SDK is available.

3-6. D-Script Engine

D-Script Engine has the role of executing D-Script safely and reliably. D-Script includes prescriptions of what to monitor and how to react to the monitoring results. It controls D-Application Manager, D-Application Monitor, and D-System Monitor.

4. Usage of D-RE

This document briefly describes the use of D-Application Manager. For installation, please refer to Appendix A1, “D-RE Installation Guide”.
For the other subsystems, such as D-System Monitor, D-Application Monitor, D-Script Engine, please refer to documents regarding each.

4-1. Usage of D-Visor

D-Visor86 and Linux KVM are available for use as D-Visor. D-Visor86 runs on a multi-core x86 processor and operates together with D-System Monitor. Please refer to "Environment Setup Manual for D-Visor + D-System Monitor" (in Japanese) concerning system requirements and installation.
On Ubuntu, packages for Linux KVM are available and are easily installed. Commands and APIs of D-Application Manager use the Linux KVM. In order to embed D-System Monitor into KVM, modification and re-build of QEMU-KVM is required. Please refer to "Environment Setup Manual for QEMU-KVM + D-System Monitor" concerning details of the modification and the usage.

4-2. Usage of D-System Monitor

In order to use D-System Monitor, D-Visor86 or modified QEMU-KVM must be used as the VMM. Please refer to "Environment Setup Manual for D-Visor + D-System Monitor" or "Environment Setup Manual for QEMU-KVM + D-System Monitor".

4-3. Usage of D-Application Manager

D-Application Manager mainly provides container functions and consists of the following commands and libraries.

4-3-1. Commands

The dre-core package installs the following commands under the /usr/sbin directory.

- **dre-sys** command
  It provides the following functions for system containers.
  - create, start, stop, or destroy a system container
  - checkpoint/restart, save/load of snapshot
  - migration of a system container
  - list of system containers
  - show status (started, stopped, etc.) of a system container
  - get information of resource usage of a system container
  - set limitation on resource usage of a system container

- **dre-app** command
  It provides the following functions for application containers.
  - create, start, stop, or destroy an application container
  - save/load of snapshot

- list of application containers
- show status (started, stopped, etc.) of an application container
- get information of resource usage of an application container
- set limitation on resource usage of an application container
- create/destroy a cgroup
- get information of resource usage of a cgroup
- set limitation on resource usage of a cgroup

For more details, please refer to documents under the /usr/share/doc/dre-core directory after installation of the dre-core package and to "D-RE Command Specification".

4-3-2. Libraries

The libdre0 package installs two libraries, **libdresys** and **libdreapp**, for C language under the /usr/lib directory. The libdre0-dev package installs header files for programming under the /usr/include directory.

● **libdresys** library

It provides the following functions for system containers.

- create, start, stop, or destroy a system container
- checkpoint/restart, save/load of snapshot
- migration of a system container
- list of system containers
- show status (started, stopped, etc.) of a system container
- get information of resource usage of a system container
- set limitation on resource usage of a system container

● **libdreapp** library

It provides the following functions for application containers.

- create, start, stop, or destroy an application container
- save/load of snapshot
- list of application containers
- show status (started, stopped, etc.) of an application container
- get information of resource usage of an application container
- set limitation on resource usage of an application coutainer
- create/destroy a cgroup
- get information of resource usage of a cgroup
- set limitation on resource usage of a cgroup

For more details, please refer to documents under the /usr/share/doc/libdre0-dev directory (after installation), "D-RE APIs Specification", and "D-RE APIs Sample Program Manual".

The libdaware0 package installs the **libdaware** library for C language under the /usr/lib directory. The libdaware0-dev package installs header files for programming under the /usr/include directory.

● **libdaware** library

It provides the following functions to develop application programs.

- set a callback function for signal handling
- support for log output

For more details, please refer to documents under the /usr/share/doc/libdaware0-dev directory (after installation), "D-RE APIs Specification", and "D-RE APIs Sample Program Manual"

The daware-mod package installs the daware kernel module and makes the special device /dev/daware available to associate process IDs inside and outside of an application container. It is a mechanism to be used when sending a signal to a process in an application container from outside of the application container.

For more details, please refer to the document under the /usr/share/doc/daware-mod directory.

4-4. Usage of D-Application Monitor

D-Application Monitor must be customized to the target system. DEOS Project developed a sample implementation applicable to a CD shopping site and released it to the public. For more details, please refer to "DEOS Programming Reference".

4-5. Usage of D-Box

DEOS Project is currently developing a sample implementation of D-Box, to be released in September, 2013.

4-6. Usage of D-Script Engine

D-Script Engine is a safe and reliable execution environment for a script language. Please refer to documents of script languages.

5. Conclusion

D-RE is a runtime environment to support activities for realizing Open Systems Dependability (OSD) based on stakeholders' agreements. In Chapter 1 of this document, we described general functions required to realize OSD. In Chapter 2, we explained the correspondence between required system functions and D-RE functions, in Chapter 3, we introduced functions of D-RE subsystems, and in Chapter 4, we described the usage of D-RE subsystems.

D-RE is still in the research and development stage. Published software is nothing more than a sample implementation. The software is provided "as is", without warranty of any kind, explicit or implied.

Appendix

A1. D-RE Installation Guide

A1-1. System Requirements

System requirements are as follows:

OS:	Ubuntu 12.04 LTS (i386/amd64)
CPU:	CPU which supports Intel VT or AMD-V
RAM:	more than 2GB (at least 512MB per system container)
Network:	DHCP address assignment must be available

A1-2. How to Get Packages

Executable modules and source code of D-RE for Ubuntu 12.04 LTS are available from the DEOS Web site (http://www.dependable-os.net/).

A1-3. Installation

The installation procedures are as follows:

A1-3-1. Settings for apt

1. Add the following two lines to the **/etc/apt/sources.list** file.

```
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. Update repositories.

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

A1-3-2. Installation of Packages

Install necessary packages as follows:

1. Minimum packages

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

2. Packages for development (optional)

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

A.1-3-3. Reboot of OS

Reboot the OS.

```
$ sudo reboot
```

A1-4. Initial Settings

A.1-4-1. Modification of Configuration File

Modify the **/etc/dre.conf** file, if necessary. (optional)

DRE_BRIDGE_IF:	bridge interface name
DRE_DEFAULT_SSHD_PORT:	default SSHD port for application containers
DRE_SYS_DEFAULT_BASE:	default base image used by system containers
DRE_SYS_DEFAULT_NETMASK:	default netmask for system containers
DRE_SYS_DEFAULT_GATEWAY:	default gateway for system containers
DRE_SYS_DEFAULT_DNS:	default DNS server for system containers
DRE_SYS_DEFAULT_MAC:	default MAC address for system containers
DRE_SYS_DEFAULT_MEMORY:	default memory size for a system container

A.1-4-2. Creation of a Base Image for System Containers

Create a KVM image which system containers use as a base image.

1. Execute the following command. It may take tens of minutes.

```
$ sudo dre-base
```

A.1-4-3. Settings for a Bridge

Bridge any network connection to the bridge interface (default is **br0**) specified in **/etc/dre.conf**.

1. Edit the **/etc/network/interfaces** file. (The following is an example to bridge **eth0** to **br0**.)

```
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. Restart the network.

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

A.1-4-4. Installation of **daware.mod** Kernel Module

Install the daware.mod kernel module. This must be done every time the kernel is updated.

1. Execute the following command.

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

This is all that is needed for installation and configuration of D-RE.

In order to use D-RE, please refer to the following documents published on the DEOS Web site (http://www.dependable-os.net/).

- D-RE APIs Specification (Project Document #: DEOS-FY2013-EA-01E)
- D-RE Commands Specification (Project Document #: DEOS-FY2013-EC-01E)
- D-RE APIs Sample Programs Manual (Project Document #: DEOS-FY2013-SP-01E)

(Blank Page)

DEOS Project