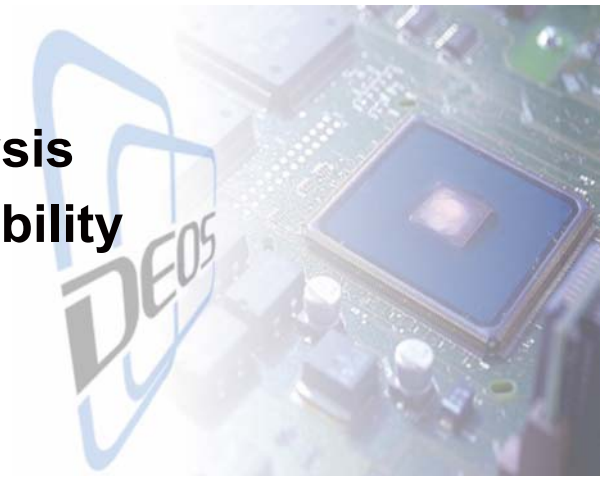


Logging and Failure Analysis for Open System Dependability

Midori Sugaya
Yokohama National University
doly@ynu.ac.jp

December 16, 2010

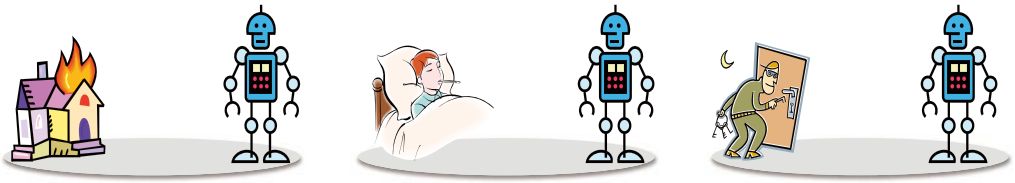
科学技術振興機構
Japan Science and Technology Agency

Team Members

- Yokohama National University
  - Midori Sugaya
  - Kimio Kuramitsu
  - Shinpei Nakata
- AIST (National Institute of Advanced Industrial Science and Technology)
  - Yoichi Ishiwata
  - Satoshi Kagami
- Dependable OS R&D Center
  - Hiroki Takamura
- Keio University
  - Satoshi Iwata

Background

- Today's advanced embedded systems (e.g. Robot) are composed of various components
- When a failure occurs,
  - various factors at stake in the system (hardware, software, applications, environments, etc..)
  - difficult to identify the root cause of the failure and accidents, especially happen in the operation phase
  - In many cases, logs are not stored as evidence
    - That is why difficult to always analyze the root cause



Disaster site　　Nursing care　　Security

Challenges and Requirements

- **Challenges : Deliver *accountability* and *visibility* by log analysis and feedback to *continuous improvements (DEOS process) for open systems dependability***

What is required log?

What kind of logs should we record?

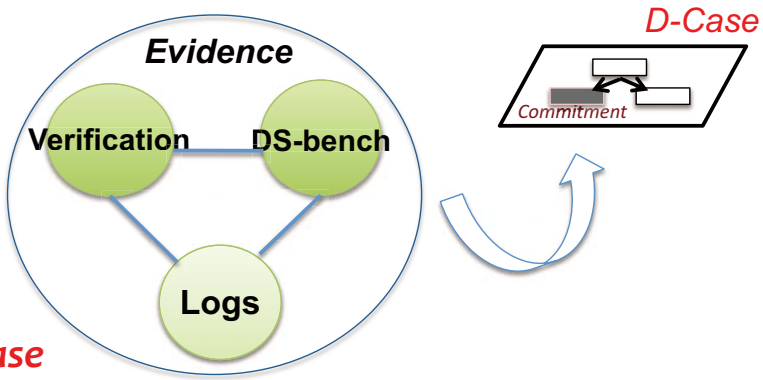
- ***Requirements***
  - ***Log* obtaining techniques to apply as an evidence for failure**
  - ***Log* analysis techniques to determine the failure**
  - **Integrating and management techniques of *logs* to store the knowledge of the failure for continuous improvements**

How much is the overhead?

Logs are sources for evidence

Evidence in DEOS Process

- Evidence is bases of an argument that support a claim of a stakeholder
  - Supported by documentations and logs

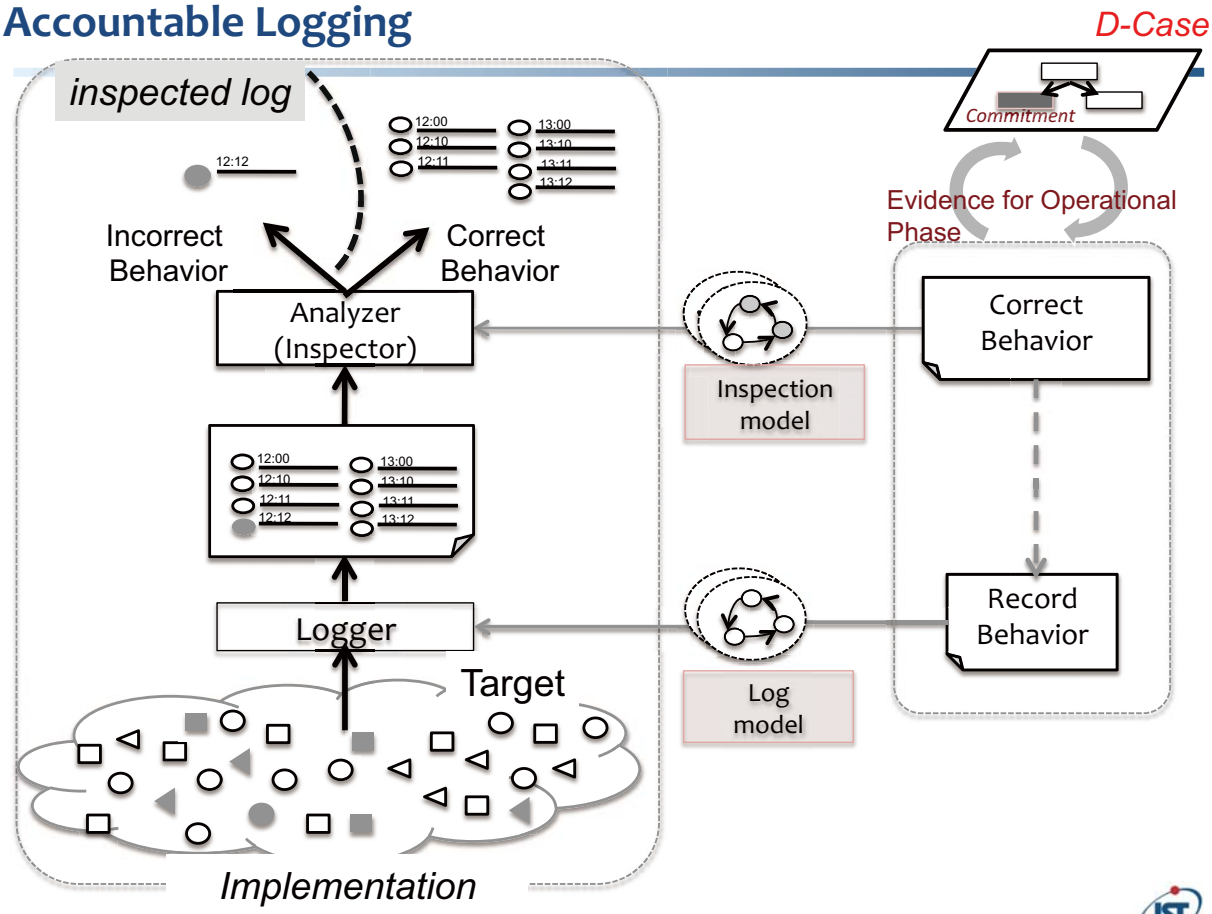


- *In System lifecycle*
  - *Development Phase*
    - Testing (DS-Bench), verification results, specifications, process of the development, documents that support the guidelines, responsive action
  - *Operation Phase*
    - Log (record), it demonstrates a system provides a service that was agreed by the stakeholders. Cause analysis results, proactive avoidance results

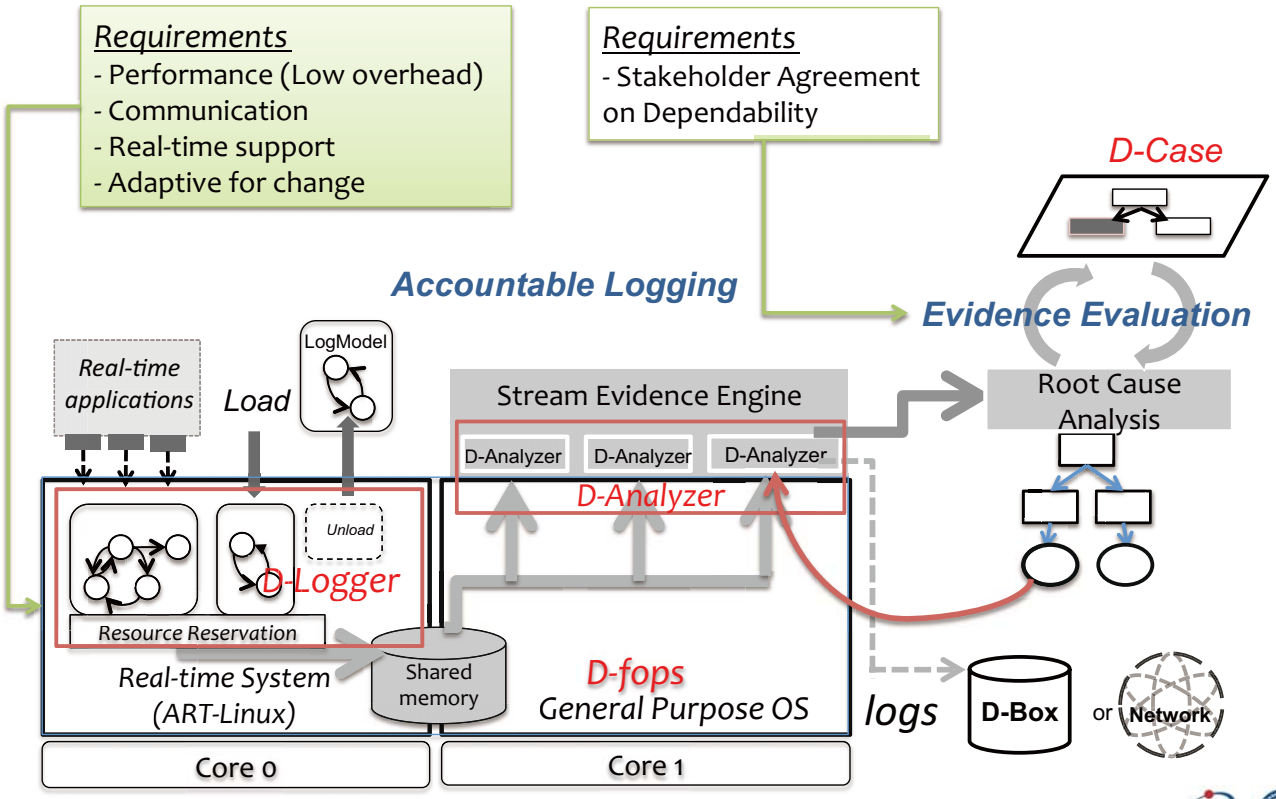
Proposal

- Accountable Logging
  - Allowing third parties to *inspect* the correctness of system behaviors *claimed* by stakeholders
    - Make an agreement upon the correct behavior of a system among the stakeholders
  - From multiple Layer by accountable logging, we collect the logs and analysis for the boundary problem.
    - User program, kernel, network, compilation information
    - We stored the *required logs* and *expected useful log* in the future
- Evidence Evaluation
  - Logs are potentially available in variety of management, root cause analysis, and feedbacks to continuous improvements

Accountable Logging



Online Logging and Analysis System

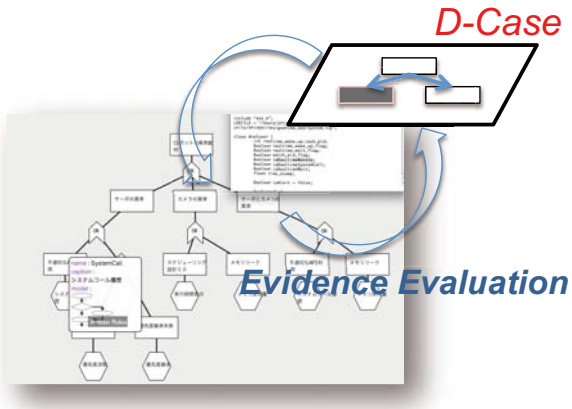


Online Logging and Analysis System Components

- *Accountable Logging*
  - *D-Logger*
    - Record the focusing events which agreed on by the D-Case.
  - *D-Analyzer (Multi-Layer)*
    - Error detection and failure analyzer is worked on the analysis core. It provides various techniques to find error and failures and their dependencies.
  - *Stream Evidence Engine (SEE)*
    - A traceable log analysis framework that includes a domain specific language for evidence analysis, and a streaming evidence analysis engine.
- *Basement System*
  - *Multi-Core based, Multi-OS Architecture*
    - Separate log analysis core to reduce the target system overhead and will not disturb real-time performance for the system.

Evaluation: Support Root Cause Analysis based on D-Case

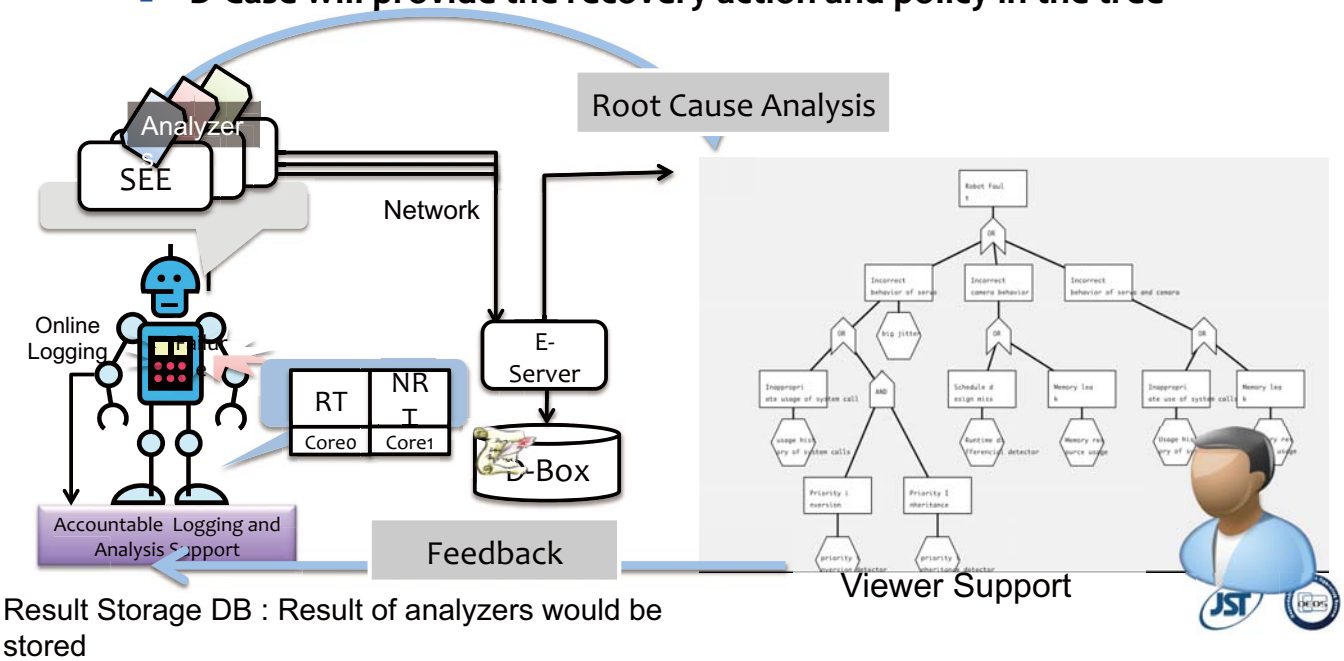
- We support root cause identification by the logging and log analysis
  - Root cause identification needs a structural understanding, solving of the problem. D-Case will give a claim for the behavior of the system



- Online-visualize method for showing problem structure whose annotation is followed as FT (FTA). It would reflect log-analysis results on FT. we call this online evolving FT as FT online.
- It provide visualizing method of our analysis results for human.
  - Human needs to understand “the problem structure”

Online support for Root Cause Analysis on Robot System Experience

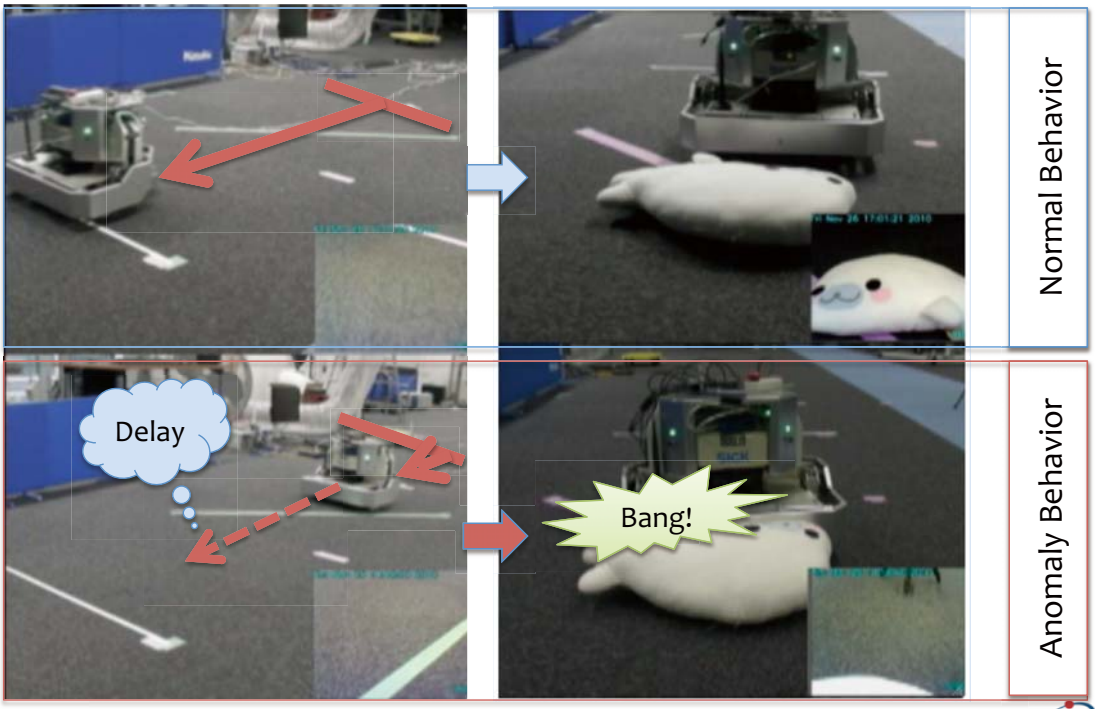
- We will provide online fault tree for showing problem structure for supporting detecting and debugging
  - It if analyzer found error in the system, it shows warnings
  - D-Case will provide the recovery action and policy in the tree



Result Storage DB : Result of analyzers would be stored

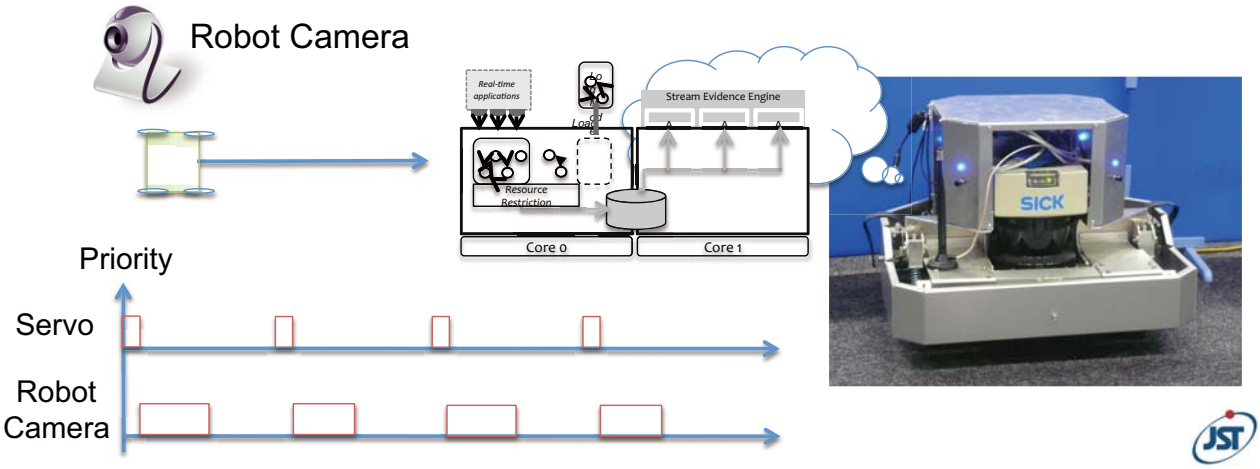
What is the Root Cause of Anomaly Behavior?

- Demonstration



Demo System Configuration

- **Basic System Architecture**
  - **ART-Linux (Provides hard real-time support for Linux)**
  - **AOS (Our proposed Accountable Logging and Analyzing System)**
- **Application**
  - **Servo Controller (Real-time Task)**
  - **Robot Camera : Detecting obstacle by image histogram analysis, It looked for around (Non Real-time Task)**



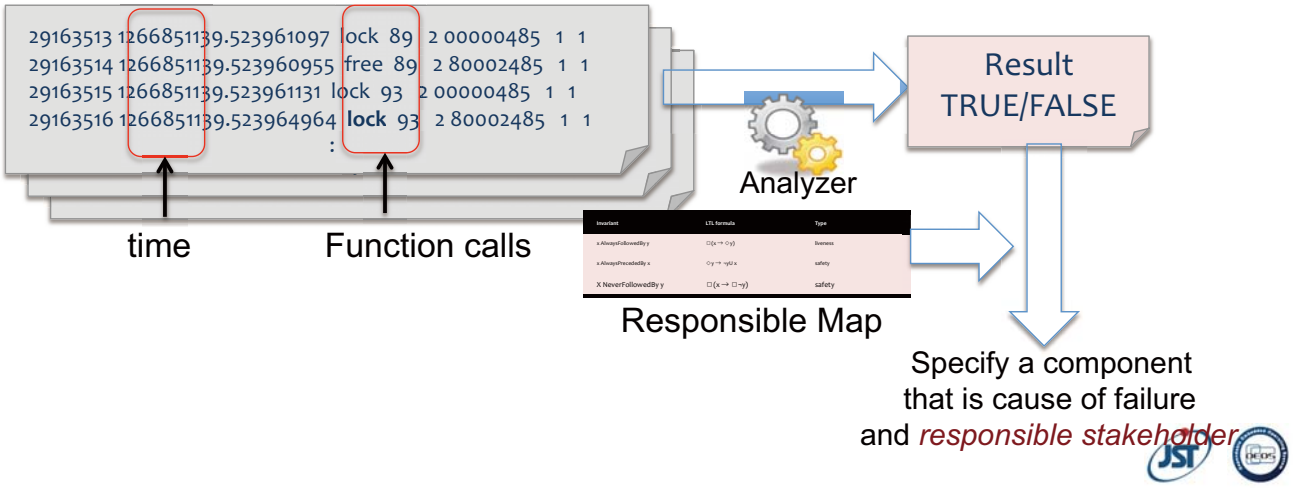
Please guess the root cause?

These are the possible answers

	Commitment	Fault
API Miss	Specification (API) art_enter, art_wait, art_exit for real-time task	Use different API (usleep) for sleep instead of art_wait
Priority Inversion	Real-time Scheduling Support, Priority Inversion should be solved by Priority Inheritance.	Developer did not understand the priority will not inherit during task blocked. There is no commitment. That cause the priority inversion
Feasibility	Real-time feasibility should be achieved by Rate Monotonic Algorithm	Scheduling Feasibility design miss (utilization is exceeded above the feasibility study)
Memory Leak	program should be implemented without any expecting bugs	A application have memory leak bug (forgot free())

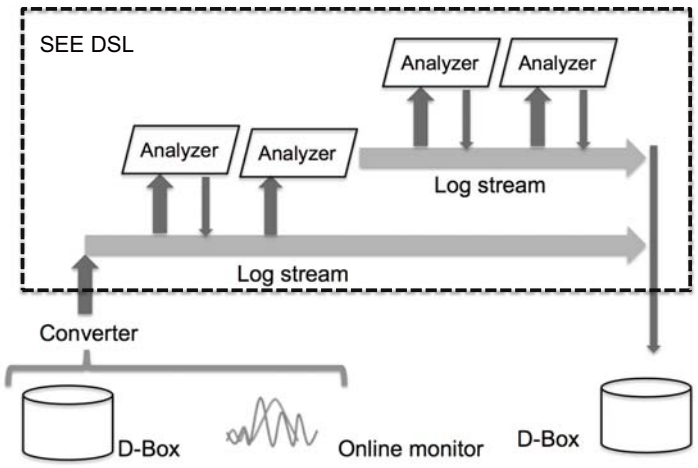
Root Cause Analysis from Log

- The execution time of the API function call was measured by collecting a time stamp just after calling API
  - In order to avoid the influence on the result of other running tasks, preemption was avoided during the measurements
- Each possibility is analyzed from log, them judged its correct or not



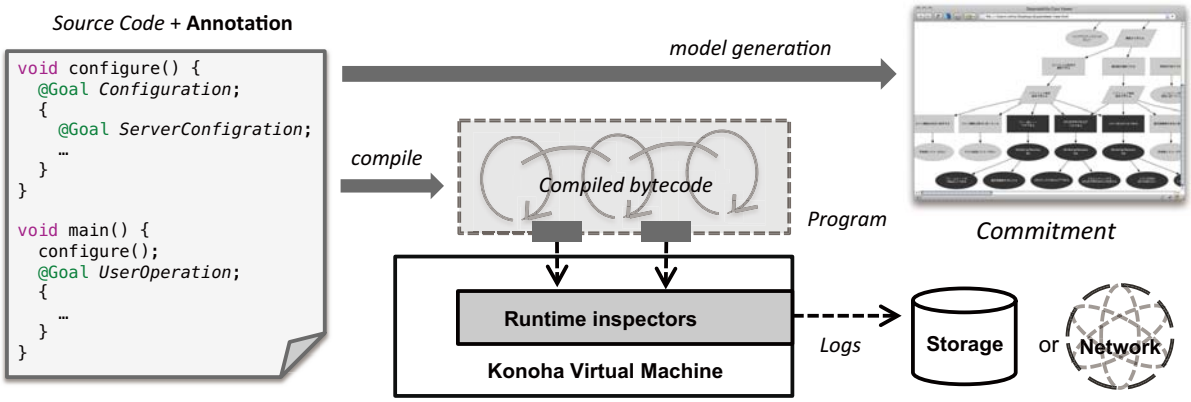
Stream Evidence Engine

- A management system that deals transparently with logs
- Provides programming framework for analysis
  - Analyzer is used to perform failure analysis, and can be handled transparently
  - Developer of analyzer simply adds an interface to SEE, and achieve the desired functionality easily



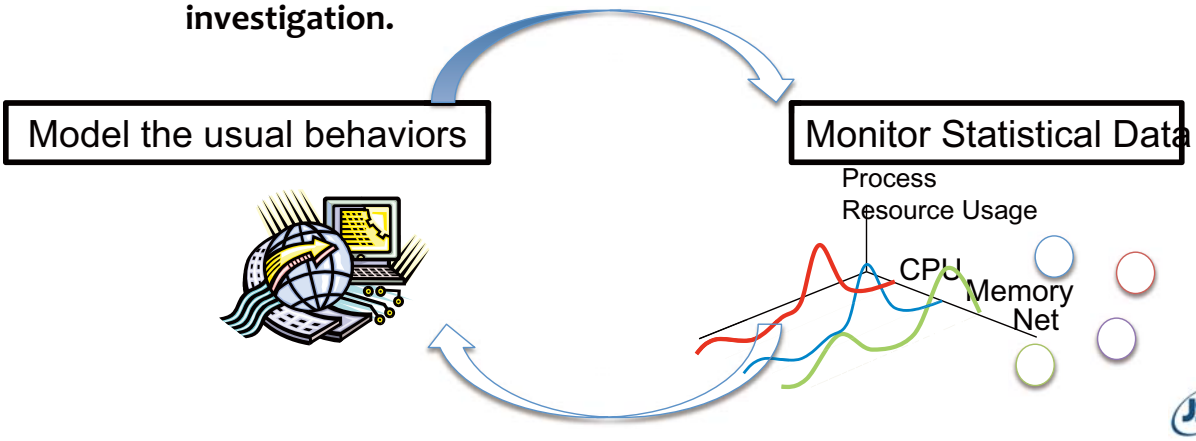
Accountable Konoha

- Model Generation form Annotation in Source Code
- Automated Logging with Language Virtual Machine
  - Model generation to create the commitment for D-Case



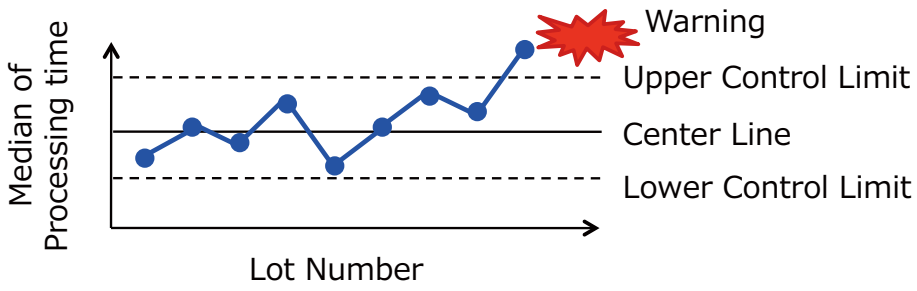
Model-Based Anomaly Detection

- Anomalistic behaviors have patterns
  - DoS attacks, fork attacks, infinite loops, …
    - Resource usages are shown the symptom of them
  - There are some symptom before failure happen
    - We will tend not to be aware of the symptom
- Modeling approach to find anomaly symptom
  - Automatically create model of normal behavior of system
  - If an anomaly model has appeared, the detector will send an alert to administrator. It would be the trigger of the further investigation.



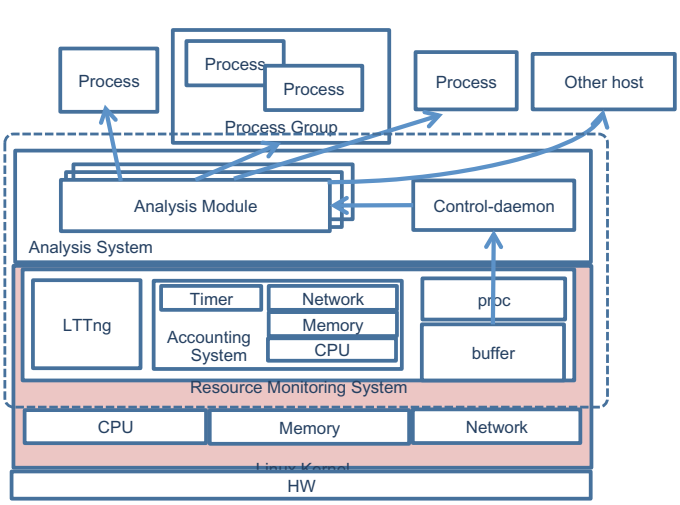
Control Charts

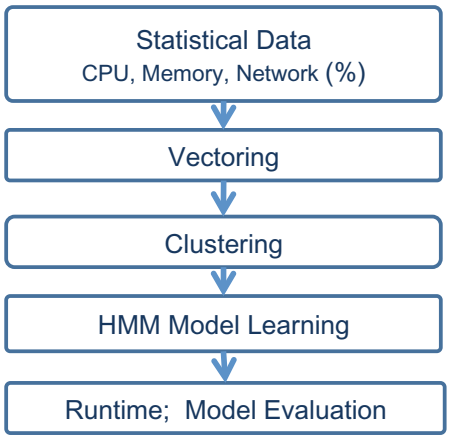
- They monitor *characteristic values* that characterize the quality of target
  - In our technique: processing time
- They raise an alarm if the measured values statistically deviate from standard values
  - To do this, they compare
    - A statistic of current characteristic values
    - Baselines calculated from the data measured in a controlled environment



Learning-based Anomaly Detection with HMM Modeling of Resource Information

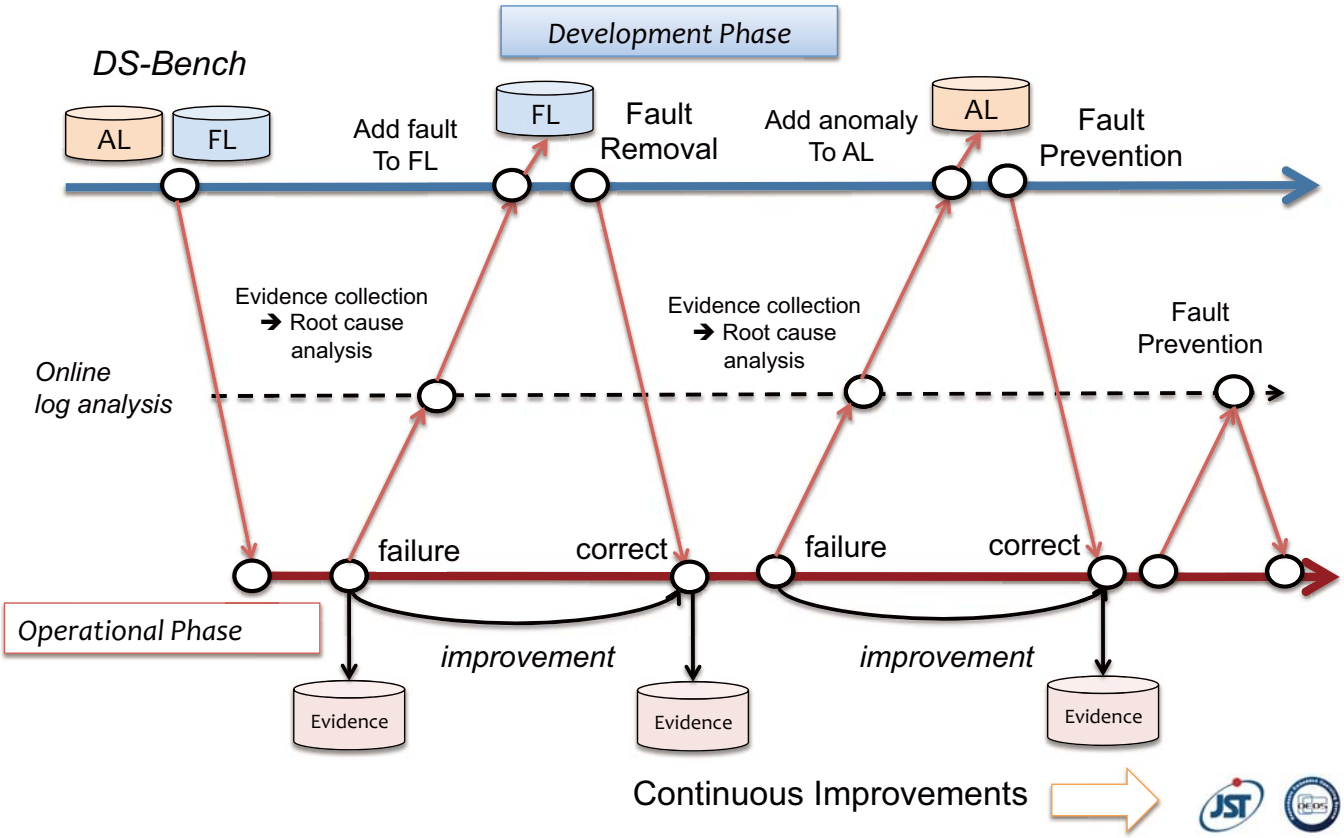
- Model-based Approach
  - A kernel level loggers: Monitor each task and collect the resource usage of them. The variable variable periods precisely used: Ayaka [Sugaya, 2009]
  - Developed Item
    - A user mode analyzer: Create models with machine learning (HMM) to find anomaly (security, and failure)





Training/Detecting Phase

Future work: Improvement Cycles



Conclusion

- We present a logging and analysis architecture and management framework for future advanced embedded system such as robotics
- Demonstration shows the difficulties to find the root cause from anomaly behavior
  - Our system and tools effectively found the root cause at this case, but the actual system is more complex and difficult to find the root cause
- In the future, we will provide a framework to store, integrate and management of the knowledge from logs as evidence
  - We need to develop feedback path from viewer and system
  - Also for open system environments, improvement cycles between development and operation phases