Academy meets Industry in Shonan
Integration of Formal Method and Testing for Model-Based Systems Engineering

OPEN SYSTEMS DEPENDABILITY AND DEOS
Achievements and Future

December 2, 2014

Mario Tokoro
Founder and Executive Advisor
Sony Computer Science Laboratories, Inc.
President, The Association of Dependability Engineering for Open Systems (DEOS Association)
Recent systems are being used for a long period of time, being modified due to changes in:

- Service objectives
- Users’ requirements
- Rules, regulation, and standards
- Technology through innovation

Systems often include legacy and off-the-shelf software. Systems exploit services provided by external systems. Systems may run on an external environment (cloud).

Whereas, demands for business continuity and accountability increase.
Consistency through lifecycle cannot be easily achieved

Various changes
- Service objectives
- Users’ requirements
- Rules, regulations, and standards
- Technology

Indeterminacy
We must admit that Recent Large Systems are *Ever-changing* in their
- Boundary
- Functions (and interfaces)
- Structures

Such Systems are considered as *Open Systems* than Closed systems
Closed Systems vs. Open Systems

Closed Systems

- The boundary of the system is definable.
- Interaction with the outer world is limited, and the system functions are fixed.
- The subsystems or components of the system are fixed and their relationship does not change over time.

Open Systems

- The boundary of the system changes over time.
- Interaction with the outer world and the system functions change over time.
- The subsystems or components of the system and their relationship change over time.
Can *absolute* dependability be achieved for Open Systems?

Admitting that we cannot achieve absolute dependability,

**Open Systems Dependability (OSD) is defined as the property of a system which has the abilities of:**

- *Accommodating requirement changes in service objectives and environment*
- *Providing continuously the services to users*
- *Supporting continuously to achieve accountability*
For the changes, the development process must incorporate *Change Accommodation Functions*

For continuous services, the operation process must incorporate *functions to remove problem factors* which may cause failures and *respond to failures* when a failure occurs in order to minimize damage.

For accountability achievement, *consensus building* among stakeholders must be performed for the development and revision of a system, and all the agreements are recorded with all the failures/sign of failures in a database.
The DEOS Process: A Process to Realize OSD

- Iterative process consists of
  - Change Accommodation Cycle to accommodate requirement changes in service objectives and environments
  - Failure Response Cycle to respond quickly and properly to failures
- Agreement Description Database (D-ADD) records all the agreements among the stakeholders for the development and operation and failures and their signs with countermeasures in order to support Accountability Achievement
How can we build consensus, describe agreements, and record agreements?

Among Stakeholders:
- End Users (Latent Stakeholder),
- Service/Product Providers (Business Owners), Systems Integrators/Designers,
- System Maintainers/Operators,
- Hardware/Communication Providers,
- Service/Product Authority, etc

Based on Assuredness
- Structured argumentation with evidence

In D-Case
- An extension of Assurance Case
Assurance for Systems Requirements is achieved by Reasonable Argumentation with Evidence.

- E.g., Safety Assurance and Dependability Assurance

- Assurance does **NOT Guarantee** but **Increase Confidence** that the Systems Requirements are/will be achieved (OMG/Structured Assurance Case Metamodel)
A Few Example Notations

Abstract Assurance Case Structure
See ISO/IEC 15026

CAE Graphical Notation
(www.adelard.com/asce/choosing-asce/cac.htm)

Graphical Notation of GSN
(www.adelard.com/asce/choosing-asce/gsn.htm)
D-Case is an argumentation method/tool extended from assurance cases to be used in the development and operation phases.

- We use the structural notation called GSN (Goal Structuring Notation) with Goal, Strategy, Context, Evidence, and Undeveloped Nodes.
- We added Monitor Node to glue the development and operation phases.
- Description in Natural Language, Pseudo (Controlled) Natural Language such as SBVR, or more formal way in Agda.

In-Operational Range of Response Time:
- 0~50ms: Normal
- 50~100ms: Severity Level 1
- 100~ms: Severity Level 2
An Example D-Case for DEOS Process

Service continuity and accountability in an ever-changing system can be achieved

Decompose by the three elements of DEOS Process

Ordinary Operation state can be achieved

• Operation rules
• Daily inspection guide
• Training plans and reports
• Operation sheets

Decompose by foreseeability

Failure Response Cycle can be achieved

• Operation rules
• Service objectives
• Advancement of technology
• Regulation, standard and environment
• Stakeholders’ requests

Decompose by monitoring targets

Change Accommodation Cycle can be achieved

• Monitoring for changes can be achieved
  • Change detection in objectives
  • Change detection in environment

• Monitoring for failures can be achieved
  • Design document for monitoring
  • Test results

• List of foreseeable failures

• Response to foreseeable failure can be achieved
  • Anomaly detection
  • Failure detection

List of unforeseeable failures

Decompose by whether planned for in design or not

Failure Response Cycle can be achieved

• Response to unforeseeable failures
  • Anomaly detection
  • Failure detection

• Recognition of unforeseeable failures

• Response to unforeseeable failures can be achieved
  • Procedural document for unforeseeable failures
  • Procedural document for accountability achievement

Reasons for exclusion

Accountability to failures not planned for in design can be achieved

• Procedural document for accountability achievement

Procedural document for accountability achievement

• Failure specific response plans
• Procedural document for accountability achievement

Procedural document for accountability achievement

Reasons for exclusion

Accountability to failures not planned for in design can be achieved

• Procedural document for accountability achievement

Procedural document for accountability achievement

• Failure specific response plans
• Procedural document for accountability achievement
D-Case Editor (Eclipse base)
Failure mitigation actions invoked.
Corresponding part of D-Case is highlighted.
How can we cope with Legacy and Off-the-shelf modules and External Services?

- Legacy and off-the-shelf modules
- External services through networks
- Systems may run in unknown environment such as clouds

- By modularization and using External Nodes
- May need reverse engineering
How can we support accountability achievement?

- Keep the history of D-Case descriptions, of logged data, and other necessary evidences in Agreement Description Database (D-ADD) for Accountability Achievement
D-ADD (Agreement Description Database) consists of three layers

- Fundamental Tools Layer
- Model Layer
- Repository Layer
Our experience shows that description for File Server System exceeds 2000 nodes of D-Case. Description for a complex system may become 10K or more.

D-Case description is renewed when accommodating changes. It is necessary to know the effect that a change in a part cause in other parts.

How to cope with indeterminacy (e.g., change in the definition of terms and context)?

Formal method cannot handle the above.

Can D-Case be read and maintained if it becomes large?

D-Case in Agda
D-Case Verifier

Graphical edit, domain-expert review using D-Case Editor

Verification, construction, generation using Agda

D-Case/Agda Extension for D-Case Verification
A Formal Assurance Case:

- It handles not only logics but ontology (vocabulary and definition).
  - Subdivision in ontology by the truth maintenance method.
  - Redefinition of ontology (openness)
  - Upward compatible with D-Case Editor

Ontology (Vocabulary and definition of terms)

- Subdivision by truth maintenance
- Redefinition for openness
- Upward compatible with D-Case Editor

Agda: a Functional programming language based on compositional type theory
(a) Ontology in a natural language-based assurance case

- Referenced documents
- Assurance argument description
- • Ontology explanation buried in referenced docs
- • Tacit ontology hidden behind uses of words without definitions
- Reader's interpretation
- Actual system, environment, achieved properties

(b) Ontology in a formal assurance case

- Theory modules that make up the theory part
- Reasoning part
- • Ontology referenced through term definitions imported from theory modules.
- • No undeclared uses of terms
- Actual system, environment, achieved properties

Validity Maintenance
- Inconsistency found → Truth Maintenance by, e.g., sub-division of concept terms
- Reality changed → Ontology altered (openness of ontology usage)
  - Belief Change by contraction / expansion of the axiom set or vocabulary

More Precisely

December 2, 2014 © Mario. Tokoro
How can we Achieve Security in Open Environment?

- By monitoring the OS kernel to check whether OS behavior is normal or not.
DEOS Architecture

- Agreement Description Database (D-ADD) which retains all the D-Case descriptions,
- Tools to support requirements management
- Tools to develop dependable software (D-DST)
- Runtime Environment to execute programs, to monitor and record the states of programs, and to respond to failures (D-RE)
DEOS Use Cases

- D-Case description for low cost satellite system by Prof. Seiko Shirasaka, Keio University
- D-Case in Agda description of a file server system with more than 2000 nodes by the team at Kanagawa University
- May companies participate in the D-Case tutorial/seminar to describe “real” problems, organized by Nagoya University and Electro-Communication University, for example
  - Automotive: Application for the development of an engine by Toyota
  - Satellite: Application for the development of small satellite system (NESTRA)
  - Robot: Application for a robot (Fuji Xerox)
  - Robot: Application for requirement definition (Denso Create)
  - And many others...
- Robot: Application of D-RE/ART-Linux for the development of biped robots at AIST
- Robot: Application of the DEOS process to the development of the MIRAIKAN guidance robot and its services by AIST
Patrol and guidance at the exhibition floor

- D-Case was described for:
  - Automatic Patrol
  - Safe Stop
  - Interaction with human operator when necessary
  - Accountability by recording various events
  - PDCA

- Number of nodes
  - Goal 66, Evidence 29, Strategy 28, Context 12,
  - Undeveloped 7, Monitor 17, Total 159
D-Case Tools

- Tool to Support Stakeholders Agreement
  -> D-Case Editor
- D-Case Editor on Web Browser
  -> D-Case Weaver
- D-Case Stencil for Power Point
  -> D-Case Stencil
- D-Case Verification Tool
  -> D-Case/Agda
- D-Case Modeling Environment
  -> D-Case OSLC

D-ADD

- D-ADD
  -> In Preparation

D-Script

- D-Script
  -> In Preparation

D-DST

- Software Verification Tools
  -> Model Checking
- Test Support Tools
  -> DS-Bench/Test-Env (DS-Bench/D-Cloud)

D-RE

- Single IP Address Cluster
  -> Dependable Single IP Address Cluster (SIAC)
- Virtual Machine Monitor and OS Monitoring Tool
  -> D-Visor + D-System Monitor
- Recorder with Falsification Detection
  -> D-Box
- System Reorder
  -> System Recorder
- DEOS Runtime Environment
  -> DEOS Runtime Environment (D-RE)
**DEOS Homepage:**

http://www.dependable-os.net/osddeos/index-e.html

---

**DEOS Welcome to Dependability Engineering for Open-changing Systems.**

**What is DEOS?**

The IT systems today must enable satisfactory services to be provided to users continuously by adapting the systems to accommodable ever-changing objectives and environments and to manage unpredictable failures. Accountability to society when failure of services has occurred needs to be achieved. This capability is called "CSD: Open Systems Dependability". DEOS is a body of knowledge needed to achieve CSD.

To realize CSD, it is to achieve service continuity and accountability for ever-changing systems. The following is indispensable:

- DEOS Process: an iterative process for achieving balances, continuous improvement.
- DEOS Architecture: an architecture that supports a DEOS Process.
- Interrelated Processes and Technologies: processes and technologies to realize individual functions.

**DEOS Architecture**

In order to apply the DEOS process to systems, an architecture that can effectively support the DEOS process is required. We think that key components of the DEOS architecture include:

- A set of tasks to support the requirements elicitation/analysis phase, and database to store statement and the process to reach agreement.
- A set of tasks for program verification, benchmarking, and fault injection test, and a suite of testing environments that monitor the system all the time, to record and report events, and to react dynamically to minimize the damage to the service when failure occurs. We named the architecture "DEOS Architecture".

---

December 2, 2014 © Mario. Tokoro
O-DA was issued as an Open Group standard in July, 2013

O-DA consists of
- Develop Assurance Cases
  - Dependability Modeling
  - Assurance Case Development
- Accountability
- Failure Response Cycle
- Change Accommodation Cycle
- with O-DA Framework, AADM, and DEOS as informative

Discussion is being done for ver.2
At IEC and ISO

- **IEC TC56 (Dependability)**
  - The concept of Open Systems Dependability was submitted to IEC TC56 in September 2012 and approved in December 2012 to start discussion at PT4.8 (to be standardized as IEC 62853), scheduled issuance by December 2016.
  - Participating as experts:
    - IEC60300-1 (Dependability management),
    - IEC62741 (Dependability case), and
    - IEC62628 (Guidance on software aspects of dependability)

- **ISO/IEC JTC1/SC7 (Software and systems engineering)**
  - ISO/IEC15026 Systems and Software Assurance (Co-editor)

At OMG

- Proposing **Machine Checkable Assurance Language**
- Proposing **Dependability Assurance Framework for Safety-Sensitive Consumer Devices**
Dependability for Open Systems (Ever-Changing Systems) will become more demanded in the near future, as ITC infrastructures proliferate.

Especially, in conjunction with a wider use of Internet of Things, Automatic Driving, Intelligent Transport Systems, ...
Assurance Cases / D-Cases are getting to be known by many people. Whereas, the importance of process and database is not well understood yet.

Publicity of OSD/DEOS
Standardization
Teaching at Universities
Promotion to wider audience i.e., to various industries including aerospace, ship, automotive, clouds operators, etc.
Collaboration with many other societies/associations are critically important
My long-term wish:

Establishing a New Scientific Methodology

Open Systems Science
Victory of Science and Technology in 20C

- Advances of Modern Sciences and Technologies have solved various problems and brought us
  - Various Industries
  - Economical Development
  - Advances in Medicine
  - Improvement of Living Standard
Never accept anything as true unless it is obviously true.

Divide each of the problems into as many parts, and work on them individually.

Solve the simplest problems first before moving on to those that are more complex.

Work step by step, and ensure that there are no omissions or errors.
Issues in 21C

- Earth Environment and Sustainability
  - Energy, Global Warming, Population, Food, Biological Diversity, Poverty and Inequality, ...
- Life and Health
- Safety
  - Global Economy, Food
  - Networked Huge Information Infrastructures
Closed Systems vs. Open Systems

**Closed Systems**

**Open Systems**
Can We Really Solve Open Systems Problems?

• *No*, in the sense that we could give strong/complete solutions to well defined (closed systems) problems.

• *Yes*, in the sense that we will be able to give a means to make the entire situation better, not worse, through our best effort.
1. Provisionally define the problem and the problem domain (i.e., system) in which the problem resides.

2. Construct a model of the system, making every effort to include all significant sub-systems.

3. Investigate whether the behavior of the model over time is self-consistent, and consistent with the actual system’s behaviors.

4. If not, revise or replace the model, and if necessary, expand, shrink or change the system.

5. Repeat until a satisfactory result is obtained.
Open Systems Science and Open Systems Dependability

1. Provisionally define the problem and the problem domain (i.e., system) in which the problem resides.

2. Construct a model of the system, making every effort to include all significant subsystems.

3. Investigate whether the behavior of the model over time is self-consistent, and consistent with the actual system’s behaviors.

4. If not, revise or replace the model (Failure Response Cycle), and if necessary, expand, shrink or change the system (Change Accommodation Cycle).

5. Repeat until a satisfactory result is obtained (overall DEOS Process).
A New Book is Coming

Open Systems Dependability
Dependability Engineering for Ever-Changing Systems
Second Edition
from CRC Press

Around 300 pages

Book cover design and the publication date is not yet decided

This is the first edition.