Consensus Building and In-operation Assurance for Service Dependability*

Yutaka Matsuno** and Shuichiro Yamamoto

Strategy Office, Information and Communication Headquarters, Nagoya University

Abstract. Recent information systems have become large and complex by interacting with each other via networks. This makes assuring dependability of systems much more difficult than ever before. For this problem, we observe that requirement elicitation and risk analysis methods should be tightly connected with assurance methods. Furthermore, requirements should be ensured also in operation in such open environment where several interdependency may exist. This paper describes our initial research result and preliminary implementation toward consensus building and in-operation assurance for service dependability. We propose a process cycle for consensus building among stakeholders with assurance cases. We extend conventional assurance cases for ensuring that stakeholders' requirements are satisfied during operation. The extended assurance case is called D-Case[16]. We also describe how D-Case is used for in-operation assurance.

1 Introduction

Recent information systems have become large and complex by interacting each other via networks. This makes assuring dependability of systems much more difficult than ever before.

For assuring dependability of such systems, we observe that stakeholders should reach consensus on dependability requirements, and there should be a mechanism to ensure that dependability requirements are satisfied during in operation in such open environment where several interdependency may exist. We extend conventional assurance cases for ensuring that stakeholders' requirements are satisfied during operation. The extended assurance case is called D-Case[16].

Based on above observation, this paper proposes a consensus building cycle which consists of the following three phases: 1) requirements elicitation and risk analysis, 2) stakeholders ' agreement on requirements, and 3) In-operation assurance using D-Case. In the course of consensus building, elicited requirements may possibly be revised. Requirements are also changing when stakeholder ' s agreements are updated(Fig.1).

* This work was done while the first author was in Information Technology Center, the University of Tokyo

** matsu@icts.nagoya-u.ac.jp

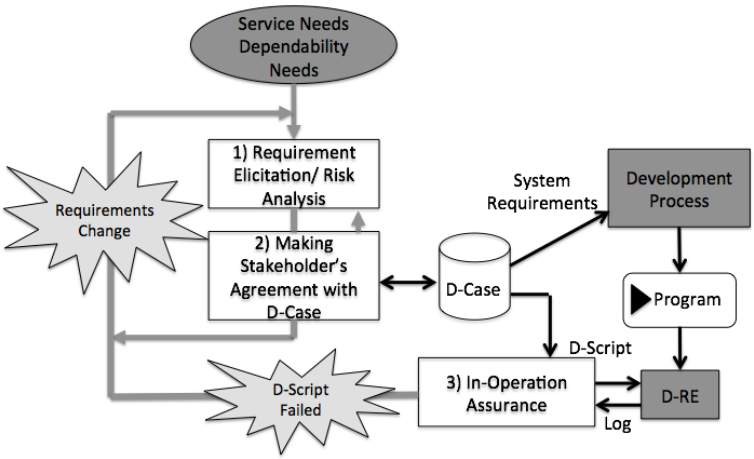

Fig. 1. Consensus Building and In-operation Assurance Cycle

Assume that a system is newly developed for some service objectives given by stakeholders. In the first phase, requirements are elicited from each of the stakeholders who have their needs described in an informal way, and then risks for their requirements are analyzed from various viewpoints. In the second phase, these elicited requirements are argued among the stakeholders using D-Case in order to reach agreement. In case the stakeholders cannot reach agreement on the requirements, some of the requirements will be returned to the first phase to revise. Once the agreement is made, programs are developed according to the D-Case description and other documents such as functional specifications. At the same time, *D-Scripts*, scripting codes for failure mitigation actions, are extracted from the D-Case description, which will be used to monitor the system, to collect logs, and to respond to failures quickly. When the system needs to be revised due to objectives/environment changes, this cycle is restarted with new requirements being elicited and old requirements being modified. This corresponds to the Change Accommodation Cycle.

The third phase provides the means to assure the agreement in the program execution by monitoring and instructing the system and managing requirements online for accountability achievement. A runtime environment called *D-RE*, monitors the system and collects logs of the system as designated by D-Scripts. If some logs show a deviation of some parameters from their in-operation ranges, the corresponding failure responsive actions designated as D-Script codes are activated. D-RE and D-Scripts have been developed in DEOS (Dependable Embedded Operating System) Project[21]. In case a need for a requirements change occurs as a result of the failure responsive actions, the above mentioned cycle is restarted with some requirements being modified. There may be a case that a failure responsive action fails to respond to the failure. Such a case may happen due to some unexpected environment changes, inadequate risk analysis, bugs of the D-Script itself, and so forth. In such a situation, the above mentioned cycle must also be restarted.

The structure of this paper is as follows. In Section 2, we introduce our requirement elicitation and risk analysis methods. Section 3 introduces D-Cases for making agreement among stakeholders and in-operation assurance. Requirements management is described in Section 4. In Section 5, we show current implementation status. Section 6 concludes this paper.

2 Requirements Elicitation and Risk Analysis

The requirements elicitation starts with the service objectives. Stakeholders can be defined according to their service objectives. Requirements are generated from each stakeholder's objectives and needs. Here, requirements include service requirements and dependability requirements. Regulations made by regulatory agencies can be considered as a kind of requirements. The activities for requirements elicitation must include identification of various levels of requirements in order for this task to be manageable.

In requirements engineering, various requirements elicitation methods have been proposed: Ethno-Methodology, Trolling, Business Modeling, Goal Oriented Analysis, Use Case Analysis, Misuse Case Analysis, Triage, etc.[7, 24, 5, 13]

We focuses on dependability in its requirements elicitation and requirements analysis. First, needs are extracted from stakeholders who describe them informally and verbally, and from these, dependability needs are obtained. Second, "dependability requirements" are identified through the analysis of dependability needs. Next, "service continuity scenarios" are created based on risk analysis and service requirements. More precisely, service continuity scenarios are developed by considering and determining countermeasures for each factor causing deviations. Finally, D-Case and D-Script are created through consensus building among stakeholders based on the service continuity scenarios.

Table 1 shows management techniques used to elicit requirements and analyze risks. Service consensus building card (SCBC) is used to define service requirements and to agree on the requirements among stakeholders. Dependability Control Board (DCB) manages consensus building process with SCBC. DCB members are representatives of stakeholders. Dependability Control Map (DCMap) describes relationships among dependability goals as well as roles of stakeholders. D-Cases are stored in D-Case DB and used to achieve dependability goals for dependability requirements of services. Service Risk Brake-down Structure (SRBS) hierarchically decomposes risks into categories. Service Fault Tree (SFT) describes the logical conditions for failures. Service Continuity Scenario (SCS) are designed to mitigate risks for dependability requirements. SCS are implemented by D-Scripts. Service Risk Management Table (SRMT) defines service risks based on probabilities and impacts of failures according to service event scenarios. Service Requirements State Management (SRSM) manages service requirements state not only during online but also offline. Fig.2 shows relationships among techniques given in Table **??**. Dependability requirements in DCMap are precisely defined and agreed on using SCBC. SRBS is then used to analyze risk category. SRMT is used to identify and mitigate risks of services

Table 1. Requirement Management Table

Techniques		Explanation
SCBC	Service consensus building card	SCBC is used to define service requirements and agree on among stakeholders
DCB	Dependability Control Board	DCB manages consensus building process with SCBC. DCB members are representatives of stakeholders.
DCMap	Dependability Control Map	DC Map describes relationships among dependability goals as well as roles of stakeholders.
D-Case DB	D-Case data base	D-Cases are stored to achieve dependability requirements of services.
SRBS	Service Risk Braek-down Structure	SRBS hierarchically decomposes risks into categories.
SFT	Service Fault Tree	SFT describes the logical conditions for failures.
SCS	Service Continuity Scenario	SCS are designed to mitigate risks to dependability requirements. SCS are implemented by D-Scripts.
SRMT	Service Risk Management Table	SRMT defines service risks based on probabilities and impacts for service event scenarios.
SRSM	Service Requirements State Management	SRSM manages service requirements state not only online but also offline.

elicited using SCBC. SFT is developed for each scenario in SRMT to show conditions of fault occurences. D-case is developed to confirm the dependability for services against risks based on the information of DCMap and SRMT. identify and mitigate risks of services elicited by SCBC. SFT is developed for each scenario in SRMT to show its occurrence condition. D-case is developed to confirm the dependability for services against risks based on the information of DCMap and SRMT.

An example of Dependability Control Map is shown in Fig. refDepControl. DCMap contains three columns that are stakeholder, roles, and dependability goals. Stakeholders and roles columns constitute RACI matrix [11]. In the role column, RACI identify roles of stakeholders such that Responsible, Accountable, Consulted, and Informed.

The dependability goals column describes goals of stakeholders and their relationships. DCMap can be used to analyze goals as follows. Users want to reach consensus on service dependability. This is accomplished by accountability achievement goal of system providers. The accountability achievement goal is supported by goals of developer and maintainer. Dependability goal of developer is also supported by hardware dependability and valid software authorization.

Table 2 shows an example of service consensus building card. SCBC consists of requirements name, event, response, input, output, functional requirements steps, initiation condition, completion condition, and roles of stakeholders. This figure omits the identification of SCBC for simplicity. Fig. 4 shows an example of Service Risk Breakdown Structure. Service risks are broken down into internal, goal, external, organizational, and technical risks. A service has a goal that is the intention and result that an actor, who wants to use the service, expects to get from the system. By getting an event from actors, services will act on objects and generate a result to achieve the goal. Services will also make responses to actors. Services work on an environment including hardware and network. Deviations of these ordinal service constituents will cause service risks. Service continuity scenarios can be constructed to mitigate these risks by considering deviations of

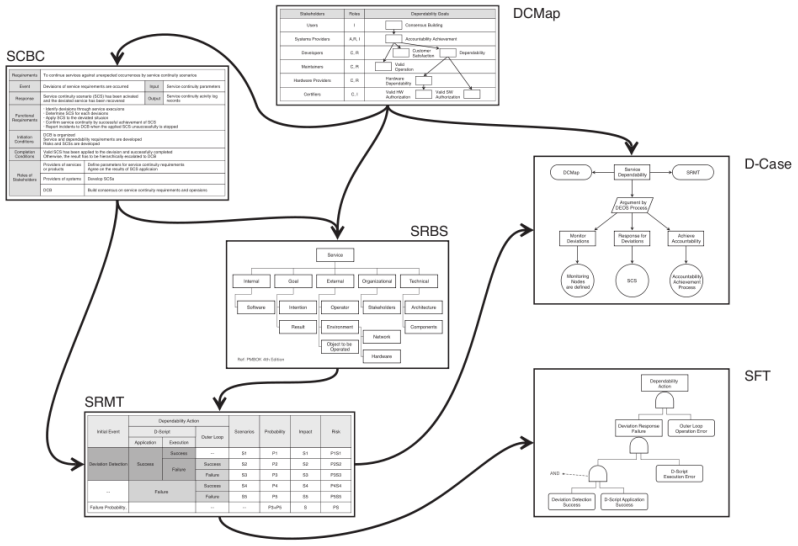

Fig. 4-2 Relationships of RM Techniques

Fig. 2. Relationship of RM Techniques

Stakeholders	Roles	Dependability Goals
Users	I	Consensus Building
Systems Providers	A,R, I	Accountability Achievement
Developers	C, R	Customer Satisfaction; Dependability
Maintainers	C, R	Valid Operation
Hardware Providers	C, R	Hardware Dependability
Certifiers	C, I	Valid HW Authorization; Valid SW Authorization

R: Responsible, A: Accountable, C: Consulted, I: Informed

Fig. 3. An Example Dependability Control Map

Table 2. An Example Service Consensus Building Card

Requirements	To keep services running after unexpected occurrences by using service continuity scenarios		
Event	Deviations from service requirements occur	Input	Service continuity parameters
Response	Service continuity scenario(SCS) is activated and the impaired service is repaired	Output	Service continuity activity log records
Functional requirements	•Identify deviations through service execution •Determine SCS for each deviation •Apply SCS to the problem situation •Confirm service continuity by successful achievement of SCS •Report incidents to DCB when the applied SCS unsuccessfully is unsuccessful		
Initiation conditions	DCB is organized Service and dependability requirements are developed Risks and SCSs are developed		
Completion conditions	Valid SCS has been applied to the deviation and successfully completed Otherwise, the result has to be hierarchically is passed on to DCB		
Roles of stakeholders	Providers of services or products	Define parameters for service continuity requirements Agree on the results of SCS application	
	Providers of systems	Develop SCSs	
	DCB	Build consensus on service continuity requirements and operations	

service constituents. This risk breakdown structure is based on those of PMBOK.

Table 3 shows an example of Service Risk Management Table. SRMT describes initial events, dependability actions, scenarios, probabilities, severity of impacts, and risks. The structure of SRMT is decomposed into two parts. The left part of SRMT describes scenarios using a binary tree of success and failure. The right part of SRMT describes the risk of each scenario.

There are two types of dependability actions. D-Scripts are applied to responsive recovery for deviations by failures. In the change accommodation cycle of DEOS process [21], human operators manage deviations in cooperation with DCB. Logical structure of failure scenarios in SRMT can be described in the similar way of fault trees.

Leveson[15] and Ericson[9] introduced methods for safety requirements analysis, such as FMEA, HAZOP, FTA, ETA. Kotonya and Sommerville showed a method for analyzing safety requirements using Hazard analysis and FTA [13]. Troubitsyna proposed Component based FMEA (Failure Mode and Effects Analysis) to analyze how component failures affect behavior of systems [22]. Sask. and Taniyama proposed Multiple Risk Communicator to the personal information leakage problem [18, 20].

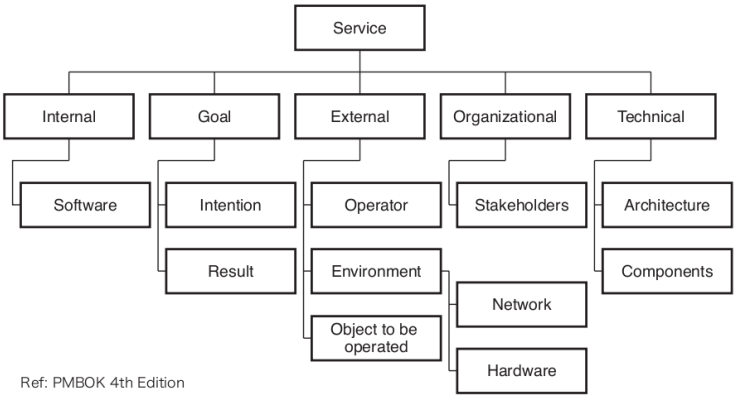

Fig. 4. An Example of Service Risk Breakdown Structure

Table 3. An Example of Service Risk Management Table

Initial event	Dependability action			Scenarios	Probability	Impact	Risk
	D-script		Outer loop				
	application	execution					
Deviation detection	Success	Success	--	S_1	P_1	S_1	P_1S_1
		Failure	Success	S_2	P_2	S_2	P_2S_2
			Failure	S_3	P_3	S_3	P_3S_3
--	Failure		Success	S_4	P_4	S_4	P_4S_4
			Failure	S_5	P_5	S_5	P_5S_5
Failure probability			--	--	P_3+P_5	S	PS

3 D-Case: Assurance Case for Stakeholders Agreement and In-Operation Assurance

It has become almost impossible to sustain dependability of the systems only by conventional methods such as formal methods and testing. We observe that the best way is stakeholders argue dependability of the system with evidences supported by experts, and try to reach agreement that the system is dependable through the whole system lifecycle. For the objectives, first, we need a method to describe and evaluate dependability requirements. Dependability requirements need to be understood by diverse stakeholders involved in the whole system lifecycle. Second, a mechanism should be in place that ensures traceability between dependability agreement and actual system behaviors. The mechanism not only keeps track of the development phases of a system, but also its run-time operations by constantly checking whether dependability requirements are being satisfied or not. In particular, we must update dependability agreement when changes occur. To achieve these two goals, we have started our study with system

assurance. The notion of assurance is to convince a person (usually to a certification body) that something is definitely true. We aim to extend assurance to agreement among stakeholders. Risk communication is used in similar contexts, but risk is only a part of dependability. We decided to exploit assurance case [4] to describe and evaluate dependability requirements. Assurance cases are structured documents for assuring dependability/safety/reliability/etc. of systems based on evidences. This simple framework has recently been widely used for safety critical domain. This is because as systems become large and complex, only following some safety checklists does not satisfy safety requirements, but assuring safety of systems becomes crucial. Assurance case is one of promising approach to dependability achievement. Current assurance cases, however, are mostly written in weakly-structured natural languages, and it is difficult to ensure traceability between assurance cases (and associated documents) and system's actual states during the whole lifecycle. Based on the above observations, we propose D-Case [16] to achieve these two goals. The two goals are re-stated as follows:

- Develop a method to evaluate and describe dependability of the system, and reach agreement among stakeholders on the dependability.
- Develop a mechanism to ensure traceability between the dependability description and the systems actual behaviors. We call this mechanism as "In-Operation Assurance".

Due to space limit, in this paper we only show our initial ideas and implementation for "In-Operation Assurance."

3.1 D-Case

Background System assurance has become very important in many industrial sectors. Safety cases (assurance cases for safety of systems) are required to be submitted to certification bodies for developing and operating safety critical systems, e. g., automotive, railway, defense, nuclear plants and sea oils. There are several standards, e.g., EUROCONTROL [10] and MoD Defence Standard 00-56, which mandate the use of safety cases. There are several definitions for assurance cases. We give one such definition as follows [1].

> a documented body of evidence that provides a convincing and valid argument that a system is adequately dependable for a given application in a given environment.

Assurance cases are often written in a graphical notation. Goal Structuring Notation (GSN) is one of such notations [12]. Writing assurance cases and reusing them in a cost effective way is a critical issue for organizations. Patterns and their supporting constructs are proposed in GSN to enable the reuse of existing assurance cases, which includes parameterized expressions. Another widely used notation is Claims, Arguments and Evidence (CAE), which was developed by Adelard and City University London [2].

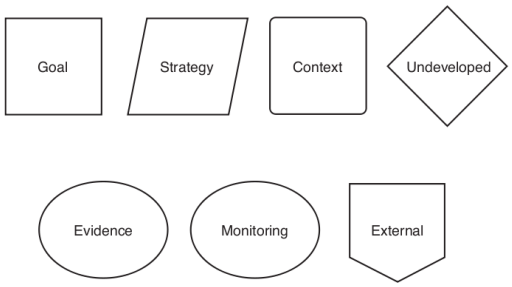

Fig. 5. D-Case Nodes

D-Case Nodes and Example Based on the assurance cases, we define D-Case. We show D-Case nodes (Fig.5) and an example (Fig.6). Current D-Case syntax is based on GSN with extensions for our needs: monitoring node and external node.

We briefly explain constructs and their meanings in D-Case. Arguments in D-Case are structured as trees with a few kinds of nodes, including: Goal nodes for claims to be argued for, Strategy nodes for reasoning steps that decompose a goal into sub-goals, and Evidence nodes for references to direct evidences that respective goals hold. Undeveloped nodes are attached to goals if there are no supporting arguments for the goals at that time. In D-Case, monitoring nodes are a sub-class of evidence nodes. They are intended to represent evidences available at runtime, corresponding to the target values of in-operation ranges. An external node is a link to the D-Case of other system. External node will be used in cases where part of the dependability of a system is supported by another system. Previously it was called "system component" node [16]. Fig. 6 is a simple example of D-Case. The root of the tree must be a goal node, called top goal, which is the claim to be argued (G1). A context node C1 is attached to complement G1. Context nodes are used to describe the context (environment) of the goal to which the context is attached. A goal node is decomposed through a strategy node S1 into sub goal nodes (G2, G3, and G4). The strategy node contains an explanation, or reason, for why the goal is achieved when the sub goals are achieved. S1 explains the way of arguing (argue over each possible fault: A and B). When successive decompositions reach a sub goal (G2) that has a direct evidence of success, an evidence node (E1) referring to the evidence is added. Here we use a result of fault tree analysis (FTA) as the evidence. The sub goal (G3) is supported by monitoring node M1. In this D-Case, G3 is supported by runtime log results. The sub goal (G4) is supported by external node (Ext1). This indicates that the dependability requirement 3 (security) in C1 would be supported by another system

3.2 In-operation Assurance

This section shows our initial idea of in-operation assurance by describing a reference implementation. A demo of our idea was presented in Embedded Tech-

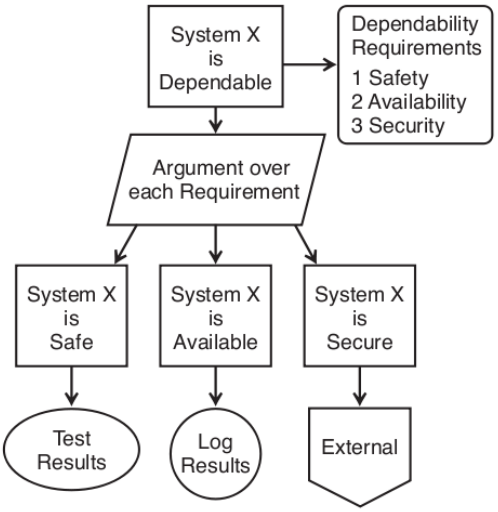

Fig. 6. D-Case Example

nology 2011, one of the largest exhibitions for embedded systems in Japan. Fig.7 shows a reference system for In-Operation Assurance.

D-Case DB contains D-Case patterns for failure response. D-Case Pattern ⇔ Module Mapping Table contains mappings between variables used in D-Case pattern and corresponding system modules. Using the table, D-Case pattern is translated to D-Script. The right-hand side of Fig.7 is a simplified D-RE, in which the Monitoring Unit and Action Unit have monitoring and failure response action modules, respectively for system components. The key concept of the reference system is that only system behaviors, which are agreed upon and stored as D-Cases, can be executed. Operators of the system would choose appropriate action as a failure response action based on D-Case from agreed upon D-Cases. Fig. 7 shows an example of D-Case pattern, which is an argument for over usage of CPU resources. The D-Case pattern argues that if CPU usage rate becomes over 50% (this can be detected by monitoring), the failure recover control unit invokes CPU resource usage module to restrict CPU usage under 50%. In Fig.8, a monitoring node is exploited. Task “A”, “CPU resource usage rate”, and “under 50%” in those monitoring nodes are value of parameters which operators and other stakeholders agreed. For example, we can specify the name of some other CPU task instead of “A”, “Memory resource usage rate” instead of “CPU usage rate”, etc. Setting the values of parameters automatically generates executable codes.

4 Requirements Management

In requirements management, as mentioned above, states of requirements are managed. There are four kinds of the states; elicited, agreed, ordinarily operated,

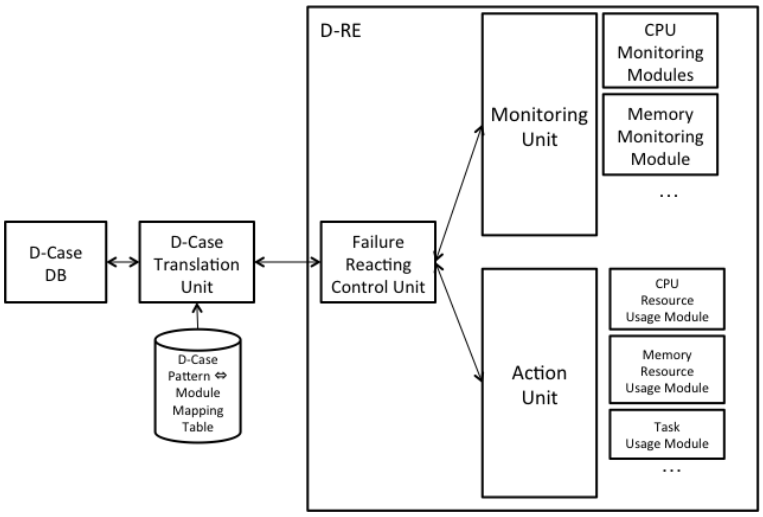

Fig. 7. A Reference System for In-Operation Assurance

and deviated (Fig. 9). First, requirements are elicited from stakeholders. These elicited requirements may conflict with each other. By consensus-building, requirements are agreed upon among the stakeholders. Agreed-upon requirements are then implemented in ordinary operations. When objectives and environments change, some ordinarily operated requirements may become obsolete and new requirements must be elicited again. This is referred to as the change accommodation cycle.

If a requirement is not fulfilled, i.e., there is deviation from the corresponding in-operation range, it moves to the deviated state. When a responsive action is possible, it moves back to the ordinarily operated state. This is referred to as the failure reaction cycle. If the service continuity scenarios cannot work for some requirements in the deviated state, these requirements should be modified and move to the elicited state. If deviations came from the implementation problems, the corresponding elicited requirements do not need any change. But it is necessary to agree on other requirements to revise the faulty implementation. This is done by consensus-building. The elicited and agreed states of requirements are managed at offline, whereas ordinarily operated and deviated states are managed online. The state of the system is represented by a set of these requirements states. Fig.10 shows how this set of requirements are managed by requirements management table as the system evolves.

Traditional requirements management (TRM) methods only consider states of requirements at offline [13, 7, 19, 14, 23, 17, 6]. These requirements engineering text books describes requirements management process by Change Control Board (CCB) with requirements change requests. TRM does not consider requirements deviations at runtime. Requirements management state model can take into account deviations of requirements at runtime. To detect and manage deviation, it is necessary to record the deviation situations on requirements with

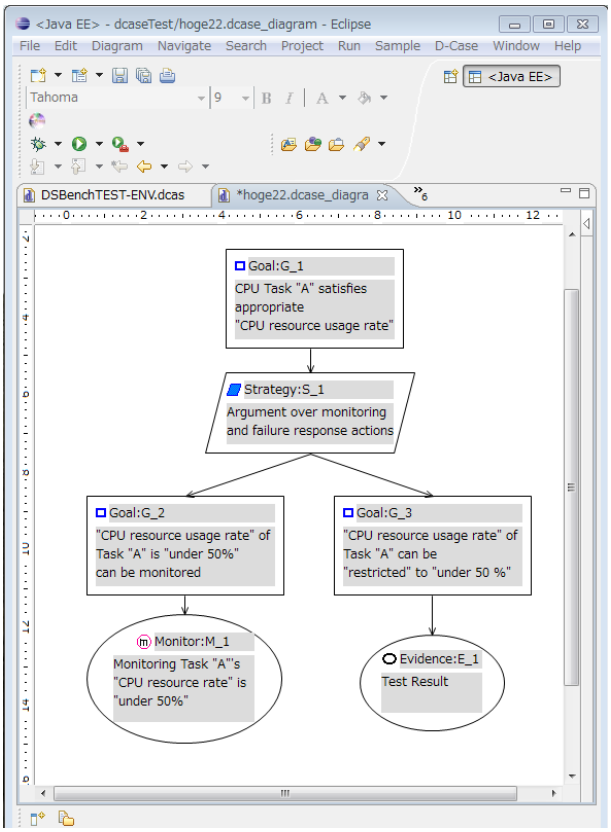

Fig. 8. An Example of D-Case Pattern

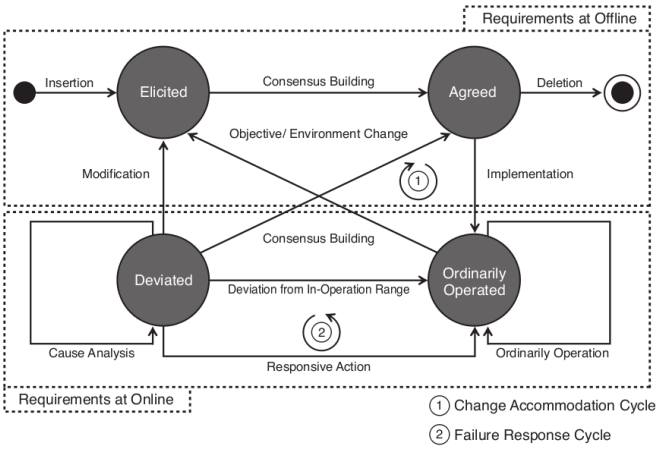

Fig. 9. Requirements State Management Model

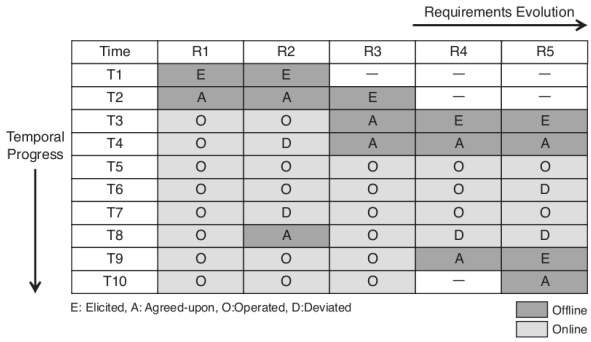

Fig. 10. System Requirement State Management Table

identifications, events, inputs, outputs, and responses. Otherwise, there is no evidence on deviations and it is impossible to analyze failures.

5 Implementation Status

Currently the requirement elicitation and risk analysis methods have been designed. Also, we have been developing "D-Case Editor" [3], which is a tool to support stakeholders' agreements, and "D-Case Viewer", which is a tool to monitor whether stakeholders' agreements are satisfied or not. Current D-Case Editor is a graphical editor as an Eclipse plug-in. Fig.8 is also a snapshot of D-Case Editor.

D-Case Editor has several basic functions and experimental functions including the followings.

1. Checks on the graph structure of D-Case (e.g. no-cycle, no-evidence directly below a strategy, etc.)
2. External documents via URL can be attached to a node.
3. "Patterns" with typed parameters can be registered and recalled with parameter instantiations.
4. Graphical diff to compare two D-Cases.
5. A "ticket" in Redmine, a project management web application, can be attached to a goal; the ticket's status can be reflected graphically in D-Case.

The main function of D-Case Viewer is monitoring: a URL to be polled by Viewer can be attached to a node; the answer is dynamically reflected in D-Case.

Fig.11 is a snapshot of web server system demo shown at ET2011, Yokohama, Japan (D-Case Viewer has been currently under development, and D-Case Editor is instead used for monitoring.) In D-Case Viewer, the monitoring node about access number of the web server and the goal are highlighted as red to indicate that access number of the web server system exceeded over 2500 times/minutes

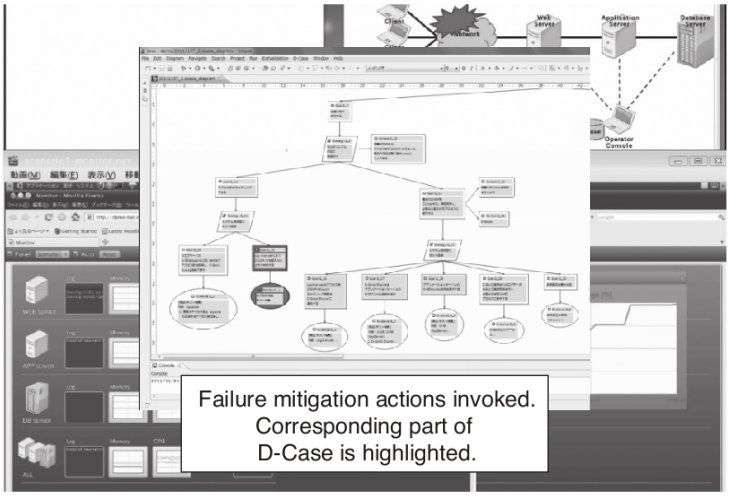

Fig. 11. A Snapshot of Web Server Demo

(this is an in-operation range). Nodes highlighted as yellow are about failure response actions invoked at just that time. Using D-Case Viewer, operators of the system can always see that all in-operation ranges are within as required or not, and which failure response actions are invoked, as agreed or not. This correspondence between D-Case description and systems actual behaviors is an important source for achieving accountability[1].

6 Concluding Remarks

This paper has reported our initial ideas and implementation for consensus building and in-operation assurance for service dependability. We have presented several methods for requirement elicitation and risk analysis. Also we have presented D-Case, which is an extension of assurance case for in-operation assurance. One clear challenge is to develop a method to describe D-Case from those requirement elicitation and risk analysis as inputs. Methods for developing assurance cases have been developed in some works such as [8]. We would like to report our progress, and compare with such works in near future.

References

1. http://www.csr.city.ac.uk/research.html.
2. http://www.adelard.com/web/hnav/ASCE/choosing-asce/cae.html.

[1] Interested reader may check a demo video: http://dl.dropbox.com/u/13455869/demo.mp4

3. http://www.il.is.s.u-tokyo.ac.jp/deos/dcase.
4. *Workshop on Assurance Cases: Best Practices,Possible Obstacles, and Future Opportunities, DSN 2004*, 2004.
5. A. Aurum and C. Wohlin, editors. *Engineering and Managing Software RequirementsEngineering and Managing Software Requirements*. Springer, 2010.
6. Brian Berenbach, Daniel Paulish, Juergen Kazmeier, and Arnold Dudorfeer. *Software and Systems Requirements Engineering In Practice*. McGraw Hill, 2009.
7. Alan Mark Davis. *Just Enough Requiremtns Management- Where Software Development Meets Marketing*. Dorset House Publishing, 2005.
8. Georgious Despotou. *Managing the Evolution of Dependability Cases for Systems of Systems*. PhD thesis, Department of Computer Science, University of York, 2007.
9. Clifton A. Ericson. *Hazard Analysis Techniques for System Safety*. John Wiley and Sons, Inc., 2005.
10. European Organisation for the Safety of Air Navigation. Safety case development manual. European Air Traffic Management, 2006.
11. IIBA. *BABOK 2.0*. 2009.
12. Tim Kelly and Rob Weaver. The goal structuring notation - a safety argument notation. In *Proc. of the Dependable Systems and Networks 2004, Workshop on Assurance Cases*, 2004.
13. Gerald Kotonya and Ian Sommerville. *Requirements Engineering-Process and Techniqeus*. John Wiley and Sons, 2002.
14. Dean Leffingwel and Don Widrig. *Managing Software Requirements A Unified Approach*. Addison-Wesley Professional, 2000.
15. Nancyy G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.
16. Yutaka Matsuno, Jin Nakazawa, Makoto Takeyama, Midori Sugaya, and Yutaka Ishikawa. Toward a language for communication among stakeholders. In *Proc. of the 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'10)*, 2010.
17. Klaus Pohl. *Requirements Enginerring Fundamentals, Principles, and Techniques*. Springer-Verlag, 2010.
18. Ryoichi Sasaki, Yuu Hidaka, Yuuko Murayama, Saneyuku Ishii, Hiroshi Yoshiura, and Hiroshi Yajima. Development concept for and trial application of a multiplex risk communicator. In *IFIP International Federation for Information Processing*, volume 189, pages 607–621. Springer, 2005.
19. Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley and Sons, 1997.
20. Mitsuhiro Taniyama, Yuu Hidaka, Masato Arai, Satoshi Kai, Hiromi Igawa, Hiroshi Yajima, and Ryoichi Sasaki. *Application of Multiple Risk Communicator to the Personal Information Leakage Problem*, pages 284–289. World Academy of Science, 2008.
21. Mario Tokoro. White paper: Dependable embedded operating system for practical use (DEOS) project, version 3, 2011.
22. Elena Troubitsyna. Elicitation and specification of safety requirements. In *ICONS 08*, pages 202–207, 2008.
23. Karl Wiegers. *Software Requirements- Practical techniques for gathering and managing requirements through the product development cycle*. Microsoft Corporation, 2003.
24. Didar Zowghi and Chad Couling. *Requirements Elicitation: A survey of Techniques, Approaches, and Tools*. Springer, 2010.