DEOS-FY2009-WP-01E

JST-CREST

Dependable Operating Systems for Embedded Systems Aiming at Practical Applications Research Area

— DEOS Project —

DEOS: Dependable Embedded Operating System

White Paper

Version 1.0

2009/09/01

DEOS Project
Research Supervisor　Mario Tokoro

Japan Science and Technology Agency

Table of Contents

1. Background

Embedded systems such as mobile information equipment, office information equipment, home appliances, and car information systems are no longer used as independent and stand alone devices, but are connected to the network, functioning as part of a large system. System services are provided to these embedded systems through the network; making these devices useful to people all over the world, bringing convenience and comfort into the lives of a ubiquitous society. However, many of these embedded systems are increasing in complexity and scale in order to meet the diverse and sophisticated needs of its users. In the development of these systems, software created by other developers which are treated as black boxes are used frequently; and often, maintenance and management of these systems involve specification changes while in use, making it impossible for developers and operators to understand every detail of the system. As such, it is becoming more difficult to ensure the reliability and availability of these embedded systems. In addition, change of external factors such as the modification of other systems that are connected via the network, fatal system problems such as crashes caused by viruses, and information leaks due to unauthorized access are happening. There is a significant increase of problems that threaten the safety and security of users; as well as the integrity and security of data. It has become a major responsibility for the product and service providers to take appropriate actions against these threats, ensuring users safety and security in continuously using these products and services. Based on the above discussion, the dependability technology that has been targeting closed systems is no longer sufficient to maintain and to improve the system's dependability; indeed, there is a need to establish a method to build and operate dependable systems that is based on new concepts and technologies [1, 4, 5, 6 and 12].

2. Dependability

2.1. Evolution of Dependability

In the late 1960's, Fault Tolerant Computer was proposed to support real-time computing and mission critical use; and an active discussion has taken place since then [20]. After which, along with the increase in scale of hardware and software and with the spread of online services, a concept called RAS, which integrates 3 properties, namely, invulnerability to failures (Reliability), maintaining a high operating ratio (Availability), and quick restoration during a malfunction (Serviceability or Maintainability), has been developed with emphasis on error detection and system recovery [8]. In the latter half of 1970's, with the addition of the preservation of data consistency (Integrity) and the prevention of unauthorized access to confidential materials (Security), systems are now being evaluated in the dimensions of RASIS—an extension of RAS. In the year 2000, the idea of Autonomic Computing was proposed to deal with complex systems connected by network to ensure dependability in the same way as the autonomic nervous system works for the human body [9, 10, 11, and 16].

Changes in the approach to reliability are reflected in international standards. International Safety Standards ISO 13849-1 (EN954-1) and Safety of Machinery - Electrical Equipment of Machines Standards IEC 60204-1 can handle simple systems, subsystems, and parts, but are not sufficient to deal with systems that include software. Functionality Safety Standards IEC 61508 was established in the year 2000 out of necessity for a safety standard for systems that include software. In IEC 61508, a system malfunction is divided into "random hardware failure" and "systematic failure".

In "random hardware failure", the failure probability is calculated from malfunctions due to deteriorating parts; while in "systematic failure", failures during system design, development, production, maintenance, and operation are made not to exceed the allowed target value through the means of a process and documentation based on the safety lifecycle, software development and verification process; such as, the V-model. The systems are categorized according to mode of

operation: low demand mode or high demand/continuous mode. The target failure limit for each mode is defined, and managed as Safety Integrity Level (SIL). The requirement level of the 4 stages from SIL1 to SIL4 is also defined (with SIL4 requiring the highest Safety Integrity level). With IEC 61058 as the base standard, machinery-related IEC 62061, process-related IEC 61511, nuclear-related IEC 61513, railway-related IEC 62278, etc. were established; and for automotives, ISO 26262 is being deliberated.

Fig.1 Dependability and Security

Efforts to produce a unified definition of dependability by integrating different ideas are continuously being done. In 1980, a joint committee of IFIP WG10.4 on "Dependable Computing and Fault Tolerance" and IEEE TC on Fault Tolerant Computing was formed, initiating a study on "The Fundamental Concepts and Terminologies of Dependability". The details and results of the following investigation have been compiled in a technical paper that was published in 2004 [2, 3]. In the said paper, dependability and security are defined as in Figure 1. However, in order to provide solutions to problems of complex modern systems, simply dividing and analyzing systems into attributes as shown in Figure 1 and dealing with each attribute separately is insufficient.

2.2. Open Systems Dependability

In order to meet the various needs of users, embedded systems have become much more sophisticated and complex; and they have grown larger in scale. Software architecture of embedded systems is determined, designed and implemented based on the requirement specifications; however, to shorten the development period and to lower development cost, the practice of using "black box" software such as existing software or software provided by other companies, has increased. Moreover, specification update for function improvement and function change occurs while the system is in operation. In this situation, fixes of the software are downloaded and new functions are added through the network. In this kind of environment, it is becoming exceedingly difficult for designers and developers to completely understand each and every detail of the system throughout the whole lifecycle (Figure 2).

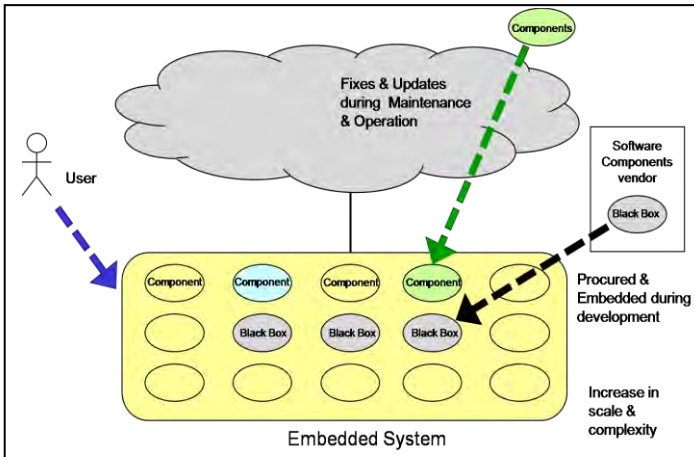

Fig.2 System Components and Services

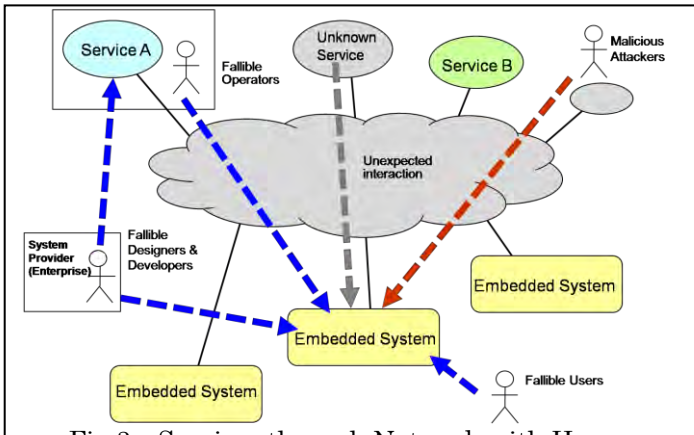

Fig.3 Services through Network with Human Interactions

Many of the modern embedded systems today are used with other systems connected via the network. In this case, users of the embedded systems utilize services through the network that a single domain (consisting of networked systems) provides. A single

service domain also connects and interacts with other domains in different levels; and, it is possible for services and interface specifications of other service domains to undergo changes; as well as discontinuation of the services themselves. In this light, the boundary of the system or service domain becomes unclear. Furthermore, there is the possibility that system designers and developers, operators and users may commit unintended errors. Likewise, along with the spread of network, unknown and unexpected interactions occur; raising the concern that the system may be attacked on purpose with malicious intents. For these reasons, predictability has increasingly become much more difficult with the advent of networking (Figure 3).

Future systems have to have the capacity to handle the issues such as those mentioned above. Handling these issues means managing changes in requirements including systems' growth through the life cycle, and changes in topology or dimension of the system; in other words, managing dependability of open systems. Until now discussions on "dependability" has focused mainly on the performance of systems that were defined as "*closed systems*". The modern dependability needs, considered as issues for open systems, are summarized as follows.

(1) Incomplete requirement specifications; difficult to fully understand as well as guarantee the system's behavior upon shipment. (Fig.4) **< Incomplete Specifications/Implementation, and the Difficulty of Understanding the System>**

- Change of stakeholders' expectations and change of items and level of requirements during design or development phase.
- Uncertainty of a component's function due to complexity, code size, usage of "black box" components, layering, maintenance of legacy code, and incompleteness of specifications of services and components that are used throughout the network.
- Inadequate understanding of the requirements, incompleteness of specifications, incompleteness of design, occurrence of implementation bug, and incompleteness of test coverage.

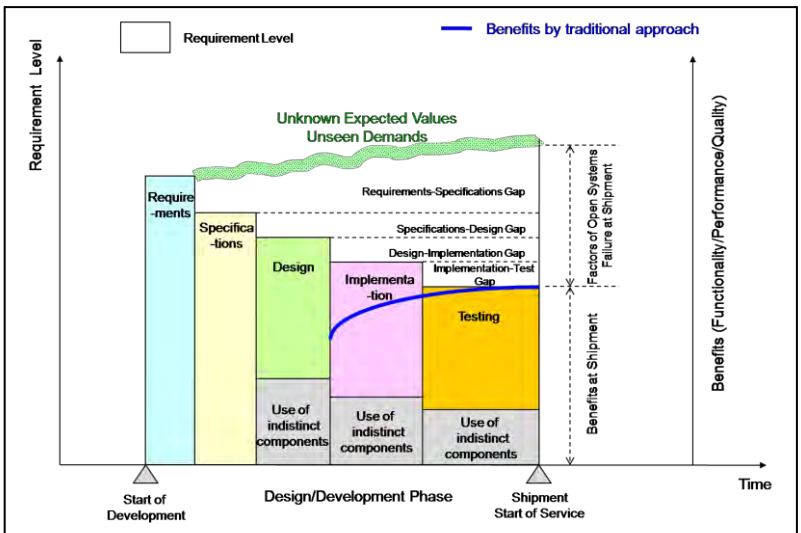

Fig.4 Incomplete Specifications/Implementation and the Difficulty of Understanding the System

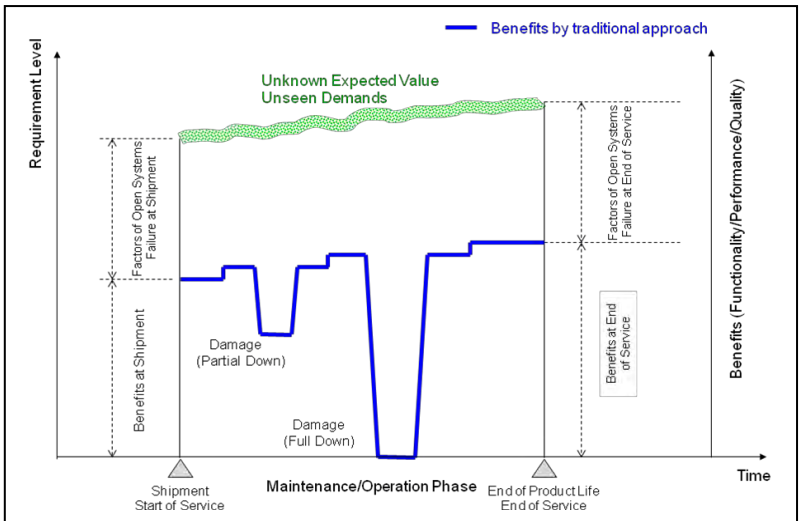

Fig.5 Uncertainty at Usage Environment, and Difficulty in Predicting System Behavior

(2) Changes in the usage environment and configuration throughout the life cycle of the system, making it difficult to completely predict the behavior of the system during the design phase. (Fig.5) **< Uncertainty at Usage Environment, and Difficulty in Predicting System Behavior>**

- Change of user's expectations or capability during maintenance or operation phase – items and level of requirements, operation capability, skill/experience/negligence.

- Complexity of managing the component configuration and the unexpected usage changes, such as those brought about by the significant increase of users or the number of shipments.
- Update or alteration of a component's function and system configuration through the network.
- Correction and update of the specifications of services and components that are being used through the network; unexpected network connections; and, intentional malicious attacks and intrusions through external entities.

It cannot be said enough that it is important for future embedded systems to be able to manage the inherent incompleteness and uncertainty of open systems. However, because of the nature of open systems, completeness in system development and operation cannot be achieved. In other words, changes of the systems to reflect changes of the user requirements cannot be avoided. We cannot, from the very beginning, design a system that can completely and properly deal with intrusions and external attacks. We cannot create a flawless system detecting beforehand all possible scenarios that could take place in the future. Failure, therefore, cannot be completely avoided.

Understanding of these situations led to various descriptions to define *"dependability"*. The following are examples of such definitions:

- *"Although the state where no failures or malfunctions occur is desirable, the ability to grasp a situation immediately when abnormalities occur, to predict the subsequent situation, and to prevent social panic and catastrophic breakdown is continuously maintained at reasonable cost."* [7]
- *"Even if various accidents have occurred, the services offered by the system are maintained at a level acceptable for the user."*[17]

However, based on the discussion we have made so far on the characteristics of modern embedded systems, we define **Open Systems Dependability** with the following description:

" Incompleteness and uncertainty could be factors that may result to failures in the future, and they are inherent to embedded systems (Factors of Open Systems Failure). Open Systems Dependability is to remove the said factors before they cause failure, to provide appropriate and quick action when they occur, to manage the failure in order to minimize the damage, and to safely and continuously provide the services expected by users as much as possible."

What is to be done to achieve open systems dependability are defined as follows:

- Provide the users with a safe and uninterrupted service by continuous effort to minimize the occurrence of failures, to minimize the damage, to recover quickly, and to prevent problems of the same cause from happening again.
- Clarify and make visible all the efforts undertaken throughout the system's life cycle from system design to development, to maintenance and operation, to modification and restoration; gathering and presenting evidence that best effort has been taken to do so. (It is imperative for the system provider to respond to the change of needs, change of external devices connected to the system, change of internal components and modules; and to explain it properly at the user's viewpoint).

"Open systems dependability" does not deny "the dependability" that has already been studied, discussed and classified so far by many researchers. Until now, with focus mainly on incidental and intentional faults, the technologies for improving the safety and security of systems have been researched, discussed and developed. Our direction is to improve the dependability of systems by minimizing the factors that cause open systems failures (mainly handled during the development phase in the manufacturer *before* shipment) and minimizing the damage of open systems failures (mainly handled during the operation phase *after* shipment), with concentration on open systems failures resulting from *incompleteness* and *uncertainty*. Indeed, "open systems dependability" complements and further enhances the concepts of "closed systems dependability".

2.3. Realizing Open Systems Dependability

There are 3 dimensions in the implementation structure that needs to be integrated in a well-organized way to properly manage open systems dependability; and they are: *elemental technologies, process & management, and system architecture* [13]. Elemental Technologies consist of technologies for system safety and implementation, technologies for design and development, as well as technologies for maintenance and operation. It specifically includes kernel extension base technologies, virtual monitor technologies, multi-core technologies, real-time or execution time prediction technologies, specification description technologies and program verification technologies. Process & Management continuously improve all processes spanning from the development of the target system to its operation. It includes standards that are necessary in ensuring accountability and compliance. System Architecture is a design specification that realizes a system, which enables effective use of elemental technologies, supports the process & management and the system's continuous progress.

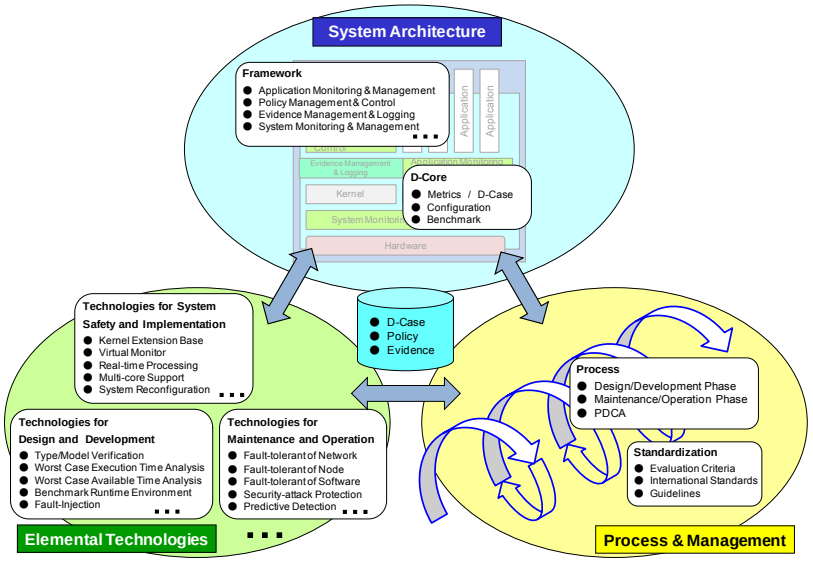

Fig.6 Realizing Open Systems Dependability

This project conceptualizes, researches, and develops this fundamental and well-organized structure, and intends to make the results widely available for embedded system providers and service providers to use (Figure 6).

3. Project Direction

3.1. Project Goal

In this project, “Dependable Operating Systems for Embedded Systems aiming for Practical Applications”, the requirements for embedded systems, such as reliability, security, and usability, are perceived from the concept of “open systems dependability”. Dependable OS for embedded systems, as well as the concepts necessary for its implementation for practical use, the architecture, specifications/implementation guidelines, management process, framework, development environment and tools and other related platform technologies are developed. Evaluation criteria are set, clarified, and standardized. In this project, “OS” is not defined as “operating system” in the strict sense of the word, but, as can be inferred from the definition of "open systems dependability", “OS” involves a broader idea, including all system software layers supporting the system applications. Moreover, our target embedded systems are defined to include systems used as social infrastructures connected to a network, such as traffic information control systems and railway ticket systems.

Note that, at present, even though systems such as monitoring systems, production control, communication control, office information equipment, vehicle information equipment, robots, information electronic devices, mobile phones, mobile information terminals, and others can be regarded as application domains that we treat as targets, further narrowing the field down according to real users’ needs can be considered.

3.2. Project Objectives

The following are the objectives of this project:

1 Establish a clear concept of dependability appropriate for the 21st century.
  1.a Evaluate and refine the open systems dependability that was discussed in Chapter 2.
  1.b Develop the elemental technologies, management process and system architecture that would enhance the open systems dependability throughout the system's life cycle.
2 Promote practical applications.
  Create specification and implementation guidelines for a dependable embedded system that can undergo actual practical usage; develop frameworks, development environment and tools; establish a management process; and provide a reference demonstration system for evaluation.
3 In accordance with 1 and 2 above, set an evaluation standard for dependability, and promote in its standardization and visualization (Establishing an international standard is expected.)
4 Start a consortium or user organization for the utilization, maintenance, and enhancement of 1 to 3 above.

The elemental technologies, process & management, and system architecture required to build a dependable system, will be evaluated and enhanced through the development of practical systems and the application of continuous improvement cycle. For this purpose, the deliverables of the project is expected to be used by the industry in the production of their products and services; subsequently improved through the feedback received on its practical usage. Furthermore, it will be essential in the future to establish a society-wide open structure to support sharing of system failure information, and to implement social responsibilities such as indemnity and accountability.

3.3. Project Deliverables

The following are the deliverables of this project:

- Open systems dependability concept
- Elemental technologies（specification for each technology, API reference document, code, implementation guidelines, etc）
- Process（process guidelines, etc）
- Framework, development environment and tools (system architecture specifications, API reference document, code, implementation guidelines, etc)
- Reference system（specifications document, code）
- Metrics, standard guidelines and record format
- Open systems dependability community
- Open systems dependability consortium

4. Items for Research & Development

4.1. Required Capability & Technology

Capability and technology to effectively manage open systems failures is essential to implement and improve "open systems dependability". Simply speaking, it is the capability and the technology *to minimize the factors of failure* and *to minimize its damage.*

◆ Capability of managing open systems failures

- Minimizing factors that cause the failures（mainly handled at the development and production site prior to shipment） （Figure 7）

1) Disclose and minimize the incompleteness (gap) in each phase (requirement, specification, design, implementation and test phase).
2) Analyze the operation of elemental technologies, and verify their behavior.
3) Record the status of operation, and support the management of product or system provider's accountability.
4) Verify conformance to International Standards.

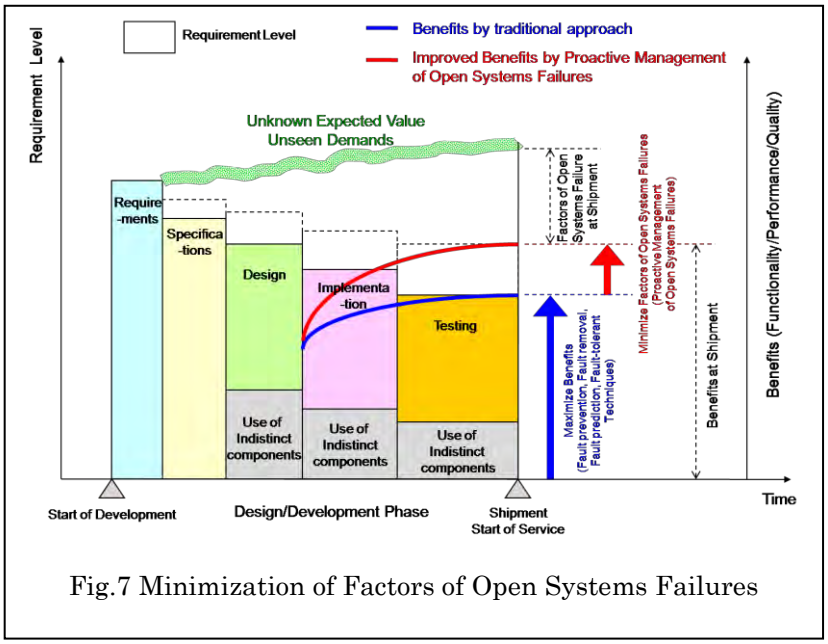

Fig.7 Minimization of Factors of Open Systems Failures

- Minimizing damage caused by failures (mainly handled at operation site after shipment) (Figure 8)

1) Temporarily operate in real-time environment, to check for possible faults and problem exposures.
2) Detect signs of failure during operation.
3) Minimize the damage when failure occurs, and perform quick system restoration.
4) Record the status of operation, and support the management of product or system provider's accountability.

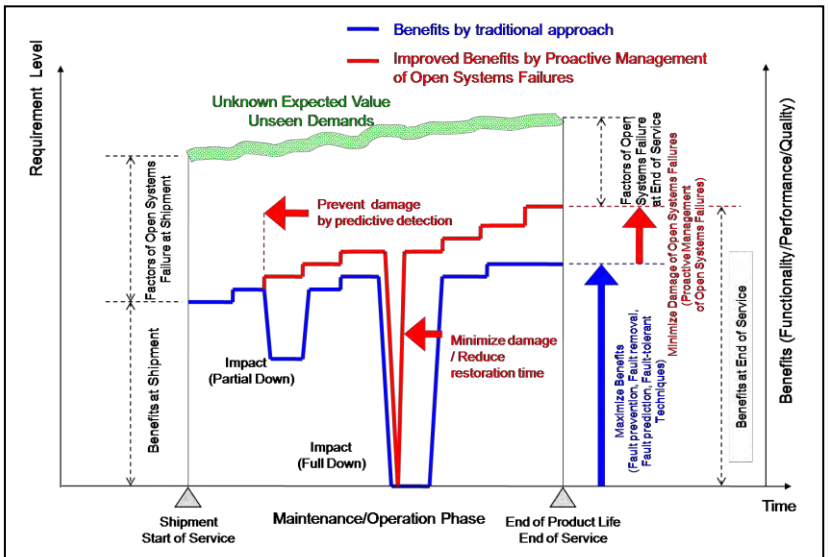

Fig.8 Minimization of Damage of Open Systems Failures

4.2. Elemental Technologies

The following are essential elemental technologies to achieve the required capability:

(1) Technology that can accurately define and verify specifications in line with the requirements
(2) Technology that can reflect the specifications to design and verify it
(3) Technology that can reliably realize the design to implementation and verify it
(4) Technology that can compare the implementation with the specification, and accurately test it
(5) Technology that can allow testing on a simulator
(6) Technology that can introduce abnormal conditions on purpose for irregular states to occur, in order to measure boundaries of the system
(7) Technology that can compare the system's normal behavior with the global standards
(8) Technology that can measure the dependability evaluation index, and verify the soundness of the evaluation criteria
(9) Technology that can monitor and record various events occurring in the system
(10) Technology that can analyze and verify the events that have occurred

(11) Policy definition & management technology
(12) Technology that can detect signs of failures
(13) Technology that can identify the causes of failures
(14) Technology that can change the configuration, isolate the affected region, and restore the system to its original configuration
(15) Technology that can restrict the system's resources and control its behavior
(16) Technology that can protect the system against attacks from other nodes when connected to the network
(17) Technology that can guarantee the system is working correctly
(18) Robust logging mechanism that has a synchronized and unalterable time system
(19) Technology that can allow changes of the system's time mechanism to enable testing on future time scenarios.
(20) Technology that can test the system life cycle by enabling the acceleration and control of time

Specifications documents, API reference documents, source codes, implementation guidelines and performance test result of each of the elemental technologies are essential for practical application.

4.3. Process & Management

Up until now, most development processes of embedded systems entails the common practice of creating a reliable development plan in advance, documenting in detail the product or system specifications, and going through the long cycle of design, implementation and verification. It has become a standard practice to perform the PDCA cycle to enhance the benefits (as well as functionality, performance, and quality) of the product or of the system. This process is quite effective for development of products or systems that are not connected to a larger network and whose specifications and behavior are quite defined and predictable at the beginning of development (closed systems).

However, as mentioned earlier, with future embedded systems, it is nearly impossible to write beforehand complete specifications, envision a complete development plan, and correctly assume all network connections. For this kind of system, the management process for developing specifications based on a perceived scope and for updating these specifications repeatedly by gathering feedback during the whole life cycle, is very crucial; as well as, the presence of elemental technologies and system architecture to enable this process [18]. In this case, using a cycle that is shorter, more flexible, and has faster turnaround time is more suitable than the usual PDCA cycle. Likewise, more careful observation and supervision of the operating conditions coupled with deeper analysis of occurring events is vital for proper management.

For practical application of this concept, process guidelines for implementation of the process as well as sample execution case studies will be needed. For accountability when incidents occur, standard management practices and guidelines for action are important and shall be discussed in the later sections.

4.4. System Architecture

System architecture plays an important role in linking elemental technologies and process & management. It is required to provide the following structures.

- Mechanism and/or structure for implementing elemental technologies
- Mechanism and/or structure for supporting the process & management (planning, execution, monitoring, and analysis cycle)
- Mechanism and/or structure for supporting system progress, and for enabling changes while the system is in operation.

In this project, this structure will be implemented and provided as a "framework " and as a "reference system". In addition, to encourage wide adaptation by system designers and providers, “specifications documents", “API reference documents" and "implementation guidelines" will also be made available. Moreover, the dependability core (D-Core) will be provided to agree on the dependability requirements between system providers and developers not only through the design and development phase, but also through the maintenance and operation phase. This is composed of a tree-structured method of describing the requirements (D-Case), and metrics, configuration, and measurement / evaluation / verification tools.

Through a business agreement between the system user and the system provider, the level of service, the functionality and other service definitions will be decided upon. For example, when the system generates an error, it is necessary to decide beforehand whether the system will notify the operator with a warning and suspend its operations, or whether the system will analyze and try to fix the error. As a system operating policy, it is necessary that this will be properly defined, administered to and executed to the system.

The configuration of a target product or system varies greatly depending on the provided service and functionality, the extent of its usage, and the level of dependability that must be maintained. Furthermore, the maintainability level of dependability changes according to whether a powerful hardware-assisted CPU is to be used or not, or whether virtualization through a VM (Virtual Machine) is to be implemented or not. The metrics (discussed in the next section) for ensuring the required level of dependability as well as the guidelines that will be used as a reference during design and implementation will be provided.

4.5. Metrics, Measurement/Evaluation/Verification Tools

To evaluate how the target system is achieving “Open Systems Dependability”, or to what extent the implementation is being done, evaluation index (metrics) that is easy to understand and that is recognized by the general public is necessary. Furthermore, tools and mechanisms to investigate, measure, and evaluate the practices of this said metrics are also required.

4.6. Standards and Guidelines

Although various standards and guidelines, such as IEC 61508, ISO 9000, and CMMI, have already been established for safety and quality assurance, these are mainly directed towards “current (conventional) dependability”. As previously stated, since it is the objective of this project to establish “Open Systems Dependability”, documentation formats and standards that regulate this concept will be defined and guidelines for its application will be provided. Furthermore, through international standardization, worldwide adaptation by parties and individuals is aspired.

When product or system related accidents or failure incidents occur during business operations, deciding who will explain to the user or to the general public, as well as how responsibility will be taken has increasingly become significant. When companies uphold accountability, the public perception is more important than how the product was made. The perception of the public whether accountability was upheld or not is one of the important criteria for evaluating dependability.

5. Research and Development Organization

This project adopted five research teams in 2006 and added four new teams in 2008. Each team will continue to research and develop until March 2012 and March 2014 respectively. At the Dependable Embedded OS Research and Development Center (DEOS R&D Center), the deliverables of the research teams, will be integrated for practical use, reconfigured to consider intellectual

property and maintenance issues, tested, assessed and packaged for practical use, and will be evaluated in collaboration with enterprises for application in actual products (Fig. 9).

Although there are currently nine teams participating in this project, their research topics alone may not be enough to produce the required elemental technologies. For instance, research on file system dependability is deemed important; however, it is not currently covered.

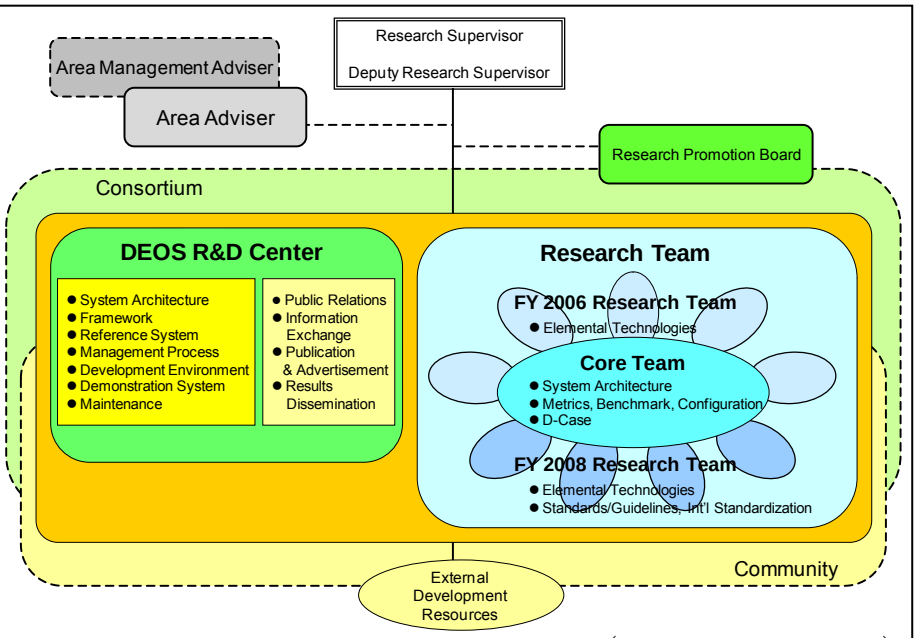

Fig.9 Research and Development Organization (As of September 2009)

From here on, through the proposal of the system architecture, and through the design (basic and detailed) of the reference system, as well as through the process of identifying the required elemental technologies, it will become clearer whether open source can be utilized as is or not; or, whether it is necessary to conduct further research and development activities for this project.

To move this project forward, Research Promotion Board members from industries are participating to represent the viewpoint of product or system providers. The board members and the product or system providers will constantly verify the requirements, the direction towards its practical implementation, and the resolution of problems related to its practical application. Furthermore, the progress of the project will be disclosed to the public, inviting opinions from people outside the team, and gathering feedbacks that can be applied to the whole project. This is done with the purpose of making the concept of dependability and the method of developing and maintaining dependable systems, common public property. The present age of borderless world of IT systems and businesses, with the advent of the internet and economic globalization, calls for the concepts and techniques in this project to be shared not only within Japan, but with the international community as well.

6. Roadmap

To fulfill the items described earlier, the following phases are the principal milestones of the entire project's progress (Fig.10).

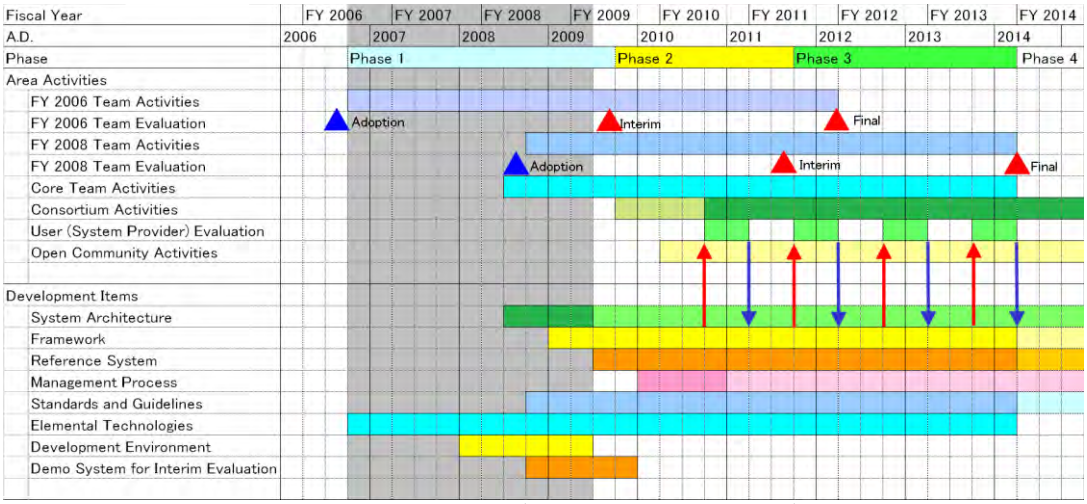

Fig.10 Roadmap (As of September 2009)

- Phase 1 (2006/10-2009/9): Establishment of the dependability concept; presentation of the system architecture containing major evaluation indexes and development/operation process supporting the said concept; and, demonstration of the 2006 research team's demo system in which a number of elemental technologies have been integrated. (The above will be presented in September 2009 by the 2006 research team during their public presentation.)
- Phase 2 (2009/10-2011/9): Implementation of the system architecture, and the "Framework" and "Reference System" which adopts the elemental technologies of the 2006 research team; establishment of a consortium (or user organization) composed of actual potential users such as those from the industry; formation of an open community for research and development of elemental technologies and system architecture; standardization of required items; trial usage of the "Framework" and the "Reference System" by consortium members, and the gathering of feedback thereof ; demonstration of the 2008 research team's "Framework" and "Reference System" where some of the elemental technologies have been integrated. (The above will be presented in September 2011 by the 2008 research team as their interim public presentation.)
- Phase 3 (20011/10-2014/3): Trial usage of the "Framework" and the "Reference System" by consortium members; the continuing evaluation and feedback; transition to actual development and commercialization; standardization of the required items.
- Phase 4 (20014/4-): Function enhancements by the open community; and continuing utilization, maintenance and development by the consortium.

7. Issues and Concerns

7.1 Handling of Intellectual Property and Copyright

The research and development deliverables of this project will be provided to as many companies and users as possible for their practical use; and the deliverables will be provided in the form of OSS (Open Source Software) as much as possible, in order to contribute to the development of a social infrastructure on dependable embedded systems. There are various kinds of OSS licensing, such as GNU GPL (GNU General Public License), GNU LGPL (Lesser General Public License), New BSD License, and MPL (Mozilla Public License). As to which license to apply, it will depend on what is considered to be the best way to spread the use of the deliverables of this project. If development is done in collaboration with enterprises, there is a possibility that necessary restrictions will be imposed according to demands of the enterprises. In this case, the best method for spreading the use of the project's future deliverables will be examined, and a policy will be decided accordingly. Furthermore, collaboration with external research and development groups, as well as standards organizations will be done as much as possible, in order to provide user-friendly deliverables. (The specifics of such collaboration will be treated as a future issue.)

7.2 Open Systems Dependability Community

In this project, in order to consolidate the knowledge related to Open Systems Dependability available all over the world, once the base system architecture has been established, concepts have been collated to some extent, and the management process has been drafted, an international community will be formed, and researchers and developers worldwide will be invited to join. In this community, discussions related to the concepts, discussions to further strengthen the architecture, discussions to present practical uses of the process, and aggressive research and development of technologies for the actual implementation or the improvement of Open Systems Dependability will be expected.

7.3 Open Systems Dependability Consortium

Enterprises and organizations who are potential users will be asked to evaluate the deliverables of this project. After receiving those evaluations, more practical deliverables will be aimed at by

running a major management process, increasing the number of endorsers and creating a consortium between potential users and DEOS R&D Center. Upon completion of this project, the said consortium will act as the mother organization focused on adding new functions and enhancements as part of maintenance as well as ensuring the continued operation of the service.

8. Appendix

8.1. Dependability Obstructions

If we try to focus on a software's working environment as a dependability obstruction, it can be divided into four categories namely, *faults from environment including operation and development*, *faults that are hardware-related*, *faults brought about by human error*, and *faults caused by a random attack*. Each fault category can be fully understood by considering the dependability obstructions in each life cycle from the time of development until termination. A summary is shown in the table below (Table 1).

	Environment (Operating Environment, Development Environment, etc.)	Hardware	Human Error	Security Issue / Risks
Specification	changes in the environment [natural environment, system environment (hardware, software), user environment (organization, operating environment), reuse of existing applications, frequent changes in requirement]	errors in hardware resource estimation, lack of consideration for software productivity	errors in specification estimation, performance simulation (inaccurate load), compliance to standards	theft of plans/specifications, sending of erroneous/malicious information from outside
Design	wrong tolerance design (for natural environment, system environment and operating environment), lack of consideration for target operator (end-user, administrator, etc.), bugs in design tool, errors in analysis of system interdependency, trouble arising from reusing existing programs, errors in design tool selection, frequent changes in specification	insufficient implementation of test functions, lack of consideration for software performance (inadequate software and hardware partitioning), specification mismatch between parts	errors in architecture selection/design, errors in interface design between modules/subsystems, escapes in design review, misinterpretation of specifications, errors in hardware performance estimation, errors in user interface design, wrong handling of exceptions, mismatch in software versions, insufficient design in fault recovery	theft of design information
Implementation and Unit testing	bugs in development tool, insufficient features of development environment, bugs in development environment (version mismatch, etc), inadequate training program, insufficient verification (failure and performance) of the software to be used, frequent changes in design	schedule delay/miss in the development of target embedded hardware, poor yield/quality, specification mismatch bugs (that can be fixed within the term/cannot be fixed within the term)	errors in coding, escapes in code review, errors in algorithm, errors in library selection, integration version mismatch, errors in timing assumption, escapes in unit tests, piracy, patent infringement	illegal production fraud, theft of source code, patent litigation, copyright lawsuit, embedding of malicious code
Integration, Test	bugs in test tool/test environment, insufficient test period, frequent changes in implementation	insufficient testing on the hardware functions, left over hardware bugs	insufficient test items (test cases, operation), errors in test result verification, errors in test items, errors in test reviews or test environment configuration	mixing of inferior parts, altering of test results, theft of test specification/results, intentional omissions
Distribution / Transportation	changes in the environment (natural environment, distribution systems, people in charge, distribution rules)	changes in environmental factors (temperature, shock and tilt during transport, submersion in water, other damages)	mixing of defective and counterfeit parts, occurrence of accidents during transport	mixing of counterfeit and stolen parts, damage, tampering
Operation	aging (changes over time), environment temperature, environment humidity, errors related to other systems' failures, system downtime due to unexpected data, input overload, shock/impact, power failures (power flicker, fluctuations, blackout, unplugging of electrical outlet), noise (electromagnetic ray, static electricity, cosmic ray), frequent version updates/patches, excessive operation cost, service termination or malfunction due to remaining bugs	deterioration of hardware (mechanical, chemical, physical), poor contact (connection, switch), excessive power consumption, noise, electromagnetic noise, heat	misunderstandings of specification, user error due to poor user interface design, incorrect operation (incorrect function selection, incorrect data entry or selection), errors in installation, errors in configuration, errors in data transfer	infiltration during operation (spyware, virus, attack), attaching of illegal modules, extraction of data, intrusion, information leakage
Maintenance, Update	irreproducible bugs/errors, bugs in maintenance/update tool , frequent (excessive) version updates, lack of version update history data, excessive maintenance cost	lack of features for collecting malfunction information, maintenance period of parts, compatibility of new parts	insufficient information of a malfunction, communication errors in malfunction information, version mismatch, backup failure, restoration failure, insufficient version upgrades (incomplete operation)	infiltration during maintenance/update (virus, attack), installation of illegal modules, theft of data
Disposal, Reuse	traces of deleted information	pollution, insufficient recycling or reuse plans	errors in deleting personal information, operation history, etc.	theft of information/parts

Table 1 Dependability Obstructions

To associate these obstructions to embedded OS and other peripheral technologies, the table can be divided into three areas:

(1) Area that should be covered by the process and/or organization adopted by the developer or service provider such as a company; as well as areas that should be covered in the system integration.
(2) Area where dependability technology is required by the other OS-related components.
(3) Area where large contribution of dependability can be expected from the OS and other peripheral technologies.

Each area is classified by yellow, sky blue, or orange in Table 1.

When the above-mentioned faults are foreseen, new techniques and technologies are needed to be developed to address the problems posed by modern embedded devices of the 21st century as well as traditional technologies such as CMMI, modern PM (Project Management) techniques, etc. required for proper performing of requirements/specifications definition to design, development and testing,. The reason lies in the fact that along with the technological advancement towards large-scale software, embedded systems are no longer considered as mere stand-alone products but forms a part of the foundation of the whole infrastructure that provides service to users through a network. Managing and training the engineers for this purpose has become crucial.

8.2. Related Standards and Organizations

Standard

- IEC 61508: Functional Safety
  http://www.iec.ch/zone/fsafety/fsafety_entry.htm
- IEC 60300-1: Dependability Management
  http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=text&searchfor=IEC+60300-1&submit=OK
- IEC 60300-2: Dependability Program Elements and Tasks
  http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=text&searchfor=IEC+60300-2&submit=OK
- ISO/IEC 12207: Software Life Cycle Processes
  http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=21208
- ISO/IEC 15288: System Life Cycle Processes
  http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43564

Process Guide

- CMMI: Capability Maturity Model® Integration http://www.sei.cmu.edu/cmmi/
- DO-178B: Software Considerations in Airborne Systems and Equipment Certification
  http://www.rtca.org/
- MISRA-C: http://www.misra-c.com/
- IEC 61713: Software dependability through the software life-cycle processes- Application guide
  http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=text&searchfor=IEC+61713&submit=OK
- IEC 62347: Guidance on system dependability specifications
  http://www.iec.ch/cgi-bin/procgi.pl/www/iecwww.p?wwwlang=E&wwwprog=sea22.p&search=text&searchfor=IEC+62347&submit=OK

Software

- SELinux: Security-Enhanced Linux http://www.nsa.gov/research/selinux/index.shtml
- AppArmor®: a Linux application security framework
  http://www.novell.com/linux/security/apparmor//
- Xen® hypervisor: the powerful open source industry standard for virtualization
  http://www.xen.org/

Related Organizations/Projects

- ISO: International Organization for Standardization http://www.iso.org/iso/home.htm
- IEC: International Electrotechnical Commission http://www.iec.ch/
- ISO/IEC JTC1: Joint ISO/IEC Technical Committee 1
  http://www.iso.org/iso/standards_development/technical_committees/list_of_iso_technical_committees/iso_technical_committee.htm?commid=45020
- IEC/TC56: Technical Committee 56: IEC Technical Committee for International Standards in the field of Dependability http://tc56.iec.ch/index-tc56.html
- OpenTC Consortium: Open Trusted Computing Consortium
- http://www.opentc.net/

- Linux-HA Project: High Availability Linux Project http://linux-ha.org/
- Carrier Grade Linux Workgroup : http://www.linuxfoundation.org/en/Carrier_Grade_Linux
- TCG: Trusted Computing Group https://www.trustedcomputinggroup.org/home
- CELF: CE Linux Forum, an international open source software development community http://www.celinuxforum.org/
- ERTOS Group: Embedded Real-Time Operating-Systems Group http://ertos.nicta.com.au/
- ARTEMIS: Advanced Research & Technology for Embedded Intelligence and Systems http://www.artemis.eu/
- CPS Program: Cyber-Physical Systems Program http://www.nsf.gov/pubs/2008/nsf08611/nsf08611.htm
- MISRA: Motor Industry Software Reliability Association http://www.misra.org.uk/
- AUTOSAR: Automotive Open System Architecture http://www.autosar.org/
- JasPar: Japan Automotive Software Platform and Architecture https://www.jaspar.jp/
- FlexRay Consortium: Consortium for the communications system for advanced automotive control applications http://www.flexray.com/
- NoTA: Network on Terminal Architecture http://www.notaworld.org/
- LIMO Foundation: Industry Consortium dedicated to Linux-based operating system for mobile devices http://www.limofoundation.org/

8.3. References

1. H. Yasuura, “On Dependability”, T. Nanya, “Concept and Issues on Dependability”, K. Iwano, “Dependability in Social Services”, in Dependability Workshop Report (in Japanese), CRDS-FY2006-WR-07, CRDS, JST, March 2007
2. http://www.dependability.org/wg10.4/
3. A. Avizienis, J.-C. Laprie, B. Randell, C. E. Landwehr,” Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Trans. On Dependable and Secure Computing, Vol.1, No. 1, Jan.-March 2004
4. T. Kikuno, “Requirements for Dependability in the 21th Century,” Speech at the Kickoff Symposium of JST/CREST Dependable Embedded OS Project, December 2006
5. M. Tokoro, “On Designing Dependable Operating Systems for Social Infrastructures,” Keynote Speech at MPSoC, Awaji Island, Japan, June 25, 2007.
6. Yasuura Hiroto, ”Dependable Computing for Social Systems”, Journal of IEICE、Vol.90, No.5, pages 399-405, May 2007
7. Kano Toshiyuki & Kikuchi Yoshihide, ”Dependable IT/Network”, NEC Technology, Vol.59, No.3, 2006, pages 6-10
8. M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow, “Reliability, Availability, and Serviceability of IBM Computer Systems: A Quarter Century of Progress”, IBM J. Res. Develop., Vol. 25, No. 5, 1981, pages 453-465
9. A. G. Ganek and T. A. Corbi, “The dawning of the autonomic computing era”, IBM Systems Journal, Vol. 42, No. 1, 2003, pages 5-18
10. “An architectural blueprint for autonomic computing, 4th edition”, IBM Autonomic Computing White Paper, June 2006
11. http://www-03.ibm.com/autonomic/
12. Mario Tokoro, ”Research and Development at Technology Maturity Stage”, Journal of IEICE, Vol.90, No.9, 2007, pages 742-744
13. Mario Tokoro and others, “Open System Science”, NTT Publishing Co., Ltd
14. H. B. Diab, A. Y. Zomaya, “Dependable Computing Systems”, Wiley-Interscience
15. G. M. Koob, C. G. Lau, “Foundations of Dependable Computing”, Kluwer Academic Publishers
16. M. C. Huebscher, J. A. McCann, “A survey of Autonomic Computing”, ACM Computing Surveys, Vol. 40, No.3, Article 7, August 2008, pages 7:1-7:28
17. Matsuda Koichi, Foreword, IPA SEC journal No.16, Volume 5, No. 1(Volume 16), 2009, page 1
18. T. Forbath, interview ”Japanese Company should break-away from the 20th century Development Process ” NIKKEI ELECTRONICS 2009.2.23, page 29
19. A. Avizienis, ” Design of fault-tolerant computers”, In Proc. 1967 Fall Joint Computer Conf., AFIPS Conf. Proc.Vol.31, pages 733-743, 1967