Definition and Realization of Software Accountability

Koichiro Ochimizu and Ryo Hayasaka

July 20, 2007
IS-RR-2007-010

School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1–1 Asahidai, Nomi, Ishikawa 923-1292, Japan
{ochimizu,ryoh}@jaist.ac.jp

ISSN 0918-7553

Definition and Realization of Software Accountability

Koichiro Ochimizu† and Ryo Hayasaka†

School of Information Science, Japan Advanced Institute of Science and Technology
Asahidai 1-1, Noumi City, Ishikawa, 923-1292 Japan

E-mail: {ochimizu, ryoh}@jaist.ac.jp

Abstract In this paper, we propose the new concept "Software Accountability". Our daily life heavily depends on various kinds of e-society systems. Therefore the system should be designed, implemented, operated and maintained to assure us that e-society systems are dependable and trustworthy. Software accountability is one of such requirements for Trustworthy e-Society. We try to define the concept, software accountability, based on the research results both in Software Engineering field and in Legal Theory. In the paper, we show the definition of software accountability, software accountability functions, software accountability tree, software accountability module in turn and then a realization mechanism of software accountability functions, presenting the architecture to combine the software accountability module with the existing information system. We also show the result of a case study by developing the course management system with software accountability functions that shows the feasibility of our approach.

1. Introduction

In recent years, the range of computerization of social systems has been rapidly expanding including e-governments/local-governments. The infrastructure of our social activities: administration; finance; medicine; transportation; education; business; are computerized, and all of them are connected by a network to form an e-society.

Our daily life heavily depends on such e-society systems. Therefore the system should be designed, implemented, operated and maintained to assure us that e-society systems are dependable and trustworthy in addition to the traditional services.

Katayama[1] proposed the five requirements for Trustworthy e-Society in the 21st century COE project "Verifiable and evolvable e-society system".

They are correctness, accountability, security, fault-tolerance and evolvability. Our research target is to define and realize Software Accountability and Ease-of-Evolution. The term Ease-of-Evolution means evolution with low cost.

In our society, there are a lot of laws, regulations and rules of some organizations we must obey. We call them **social rules**. In general, a social rule is a document described in some natural language, consisting of articles, clauses or sections.

An e-society system should support the application of some specific social rule, and such a system should be constructed to satisfy the social rule fully. Moreover, we need to certify and confirm that the e-society system is made satisfying the corresponding social rule correctly. Our society is always evolving, and social rules should be changed to catch up the evolution. An e-society system should evolve immediately after the change of a social rule. So we should be able to evolve the e-society system with low cost.

We call e-society systems that have the features mentioned above **Law-Enforcing Information Systems** (abbreviated as **LEIS**).

A social rule includes: the purpose of the rule; definition of legal terms; fact description; work-flow description; definition of constraints and conditions; equations to calculate something; data definition. All of them are effective in the target domain of the rule. In general, an e-society system relates part of those described in a social rule. LEIS can be classified into two categories, work-flow type and constraint-imposing type.

2. On Software Accountability

Accountability is defined as "Responsibility for the effects of one's actions and willing to explain or be criticized for them. For examples: Managers must be accountable for their decisions; the country will be held accountable for its treatment of American diplomats; corporate management is accountable to the company's shareholders."[2].

Here, we define "**Software Accountability**"[3] intuitively based on the above description.

"LEIS itself can explain the reason why it made such decisions or calculations to the user of the system. In other word, LEIS should answer the question from the stakeholders of LEIS who have some doubts about the decisions or value of calculation made by LEIS. LEIS need to make an answer using its execution histories to satisfy the stakeholders.

We used the word stakeholder to represent: people who made the social rule; people who developed the LEIS; people who operate the LEIS to show output of the LEIS; people who received the results from the LEIS. We show some examples of LEISs and their stakeholders below.

- Curriculum of some university includes rules for qualifications for completion based on its idea of education. It shows the conditions to be cleared to get qualifications. Course Management system is a LEIS that supports students to register subjects to be studied and show that how is the state of their progress. Students, teachers, administrators, system developers are typical types of stakeholders.
- Local government has a lot of regulations. There are several types of stakeholders such as lawmakers, system developer, civil service , and citizens.
- A company has a lot of in-house regulations based on its management policy. There are several types of stakeholders such as managers, and employees.

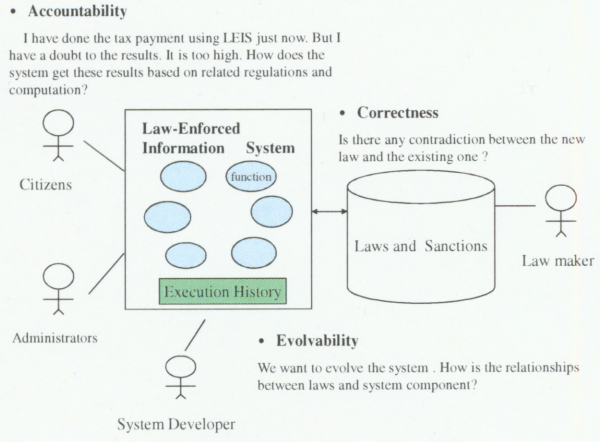

Fig. 1 Three requirements of trustworthy e-society related to our projects

We show four types of stakeholders and their typical questions to the LEIS of an e-local-government system (Fig.1).

- The lawmaker of a local government is interested in the integrity between the new law and the existing laws in addition to the content of the new law when they establish the new law. This is one of requirements of trustworthy e-society, correctness. Here a stakeholder may have the question that is "Is there any contradiction between new law and existing one?"
- The system developers are originally interested in exactly reflecting all of the content of a law into the system when they are engaged in software development. At the time, they may have questions such that "Does our system satisfy the law fully and exactly?" When they must maintain the system after law evolution, they may have a question that "We want to evolve the system. How is the correspondence between laws and system components?" That means software accountability strongly relates to the issue of evolvability of LEIS.
- Civil service and citizens are interested in the result of system execution. They may have a question that "We have done the e-application and e-registration of some information using an e-society system, for an example, a tax payment, but I have a doubt to the result shown by the system. It is too high. How does the system get these results based on what regulations and computation?".

We classify those questions of stakeholders into three categories based on the viewpoint that how does each stakeholders' question relate to what aspects of the LEIS. In Fig. 2,

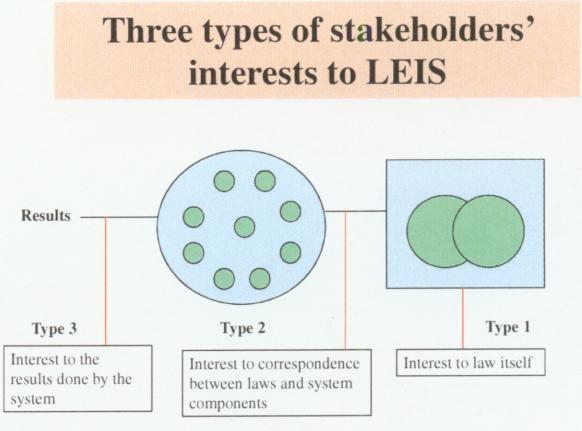

Fig. 2 Three types of stakeholders' questions

- Type 1 question relates to the social rule itself.
- Type 2 question relates to the correspondence between the social rule and the system components.
- Type 3 question relates to the execution results of the system.

Based on the problem setting mentioned above, we will discuss the following issues and will show our solution in the following sections.

In section 3, we will make clear our position to define the new concept, software accountability, by considering the issue both from software engineering standpoint and legal theory standpoint. In section 4, we will discuss how to implement the software accountability functions. In section 5, we will discuss the software architecture that enables us to attach software accountability module to the existing LEIS. And we will show the realization of software accountability functions especially for type 3 through the case study of a course management system.

3. Defining the Software Accountability

In this section, we will examine the related research results both on software engineering and legal theory to make our position clear and to define the brand new idea "software accountability"

3.1 Consideration from the Software Engineering standpoint

The research results on Requirement Engineering in Software Engineering field heavily relate to our current interests. We recognized the requirement elicitation phase as follows;

"All of the stakeholders have their own interests to their domain and LEIS directly or indirectly. They finally arrive at the understanding of the targeted world through learning efforts of the world and express the results in their own languages†."

According to the above understanding, we make clear our position on defining software accountability and we try to define what the software accountability is.

" Each stakeholder

- **has his/her own interest in the social rule and/or the system and**
- **learns and understand the related world and**
- **represents the result of learning in his/her own language before the social rule is established and the system is developed and**
- **have interest in the learning result again and**
- **try to acquire the learning result again after the social rule is established and the system is developed (Fig. 3) "**

---

† This understanding of requirement elicitation phase was obtained through the private discussion with Mr. Akira Kumagai of Tokyo Electron Software technologies.

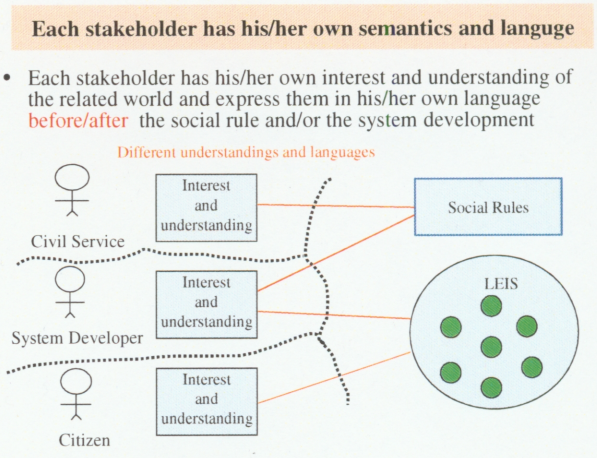

Fig. 3 Interest and understanding of each stakeholder should be recorded and shared by stakeholders

These learning results should be shared by the system and can be retrieved if needed to realize the software accountability.

From this viewpoint, traditional requirement engineering approach has some problem (Fig. 4).

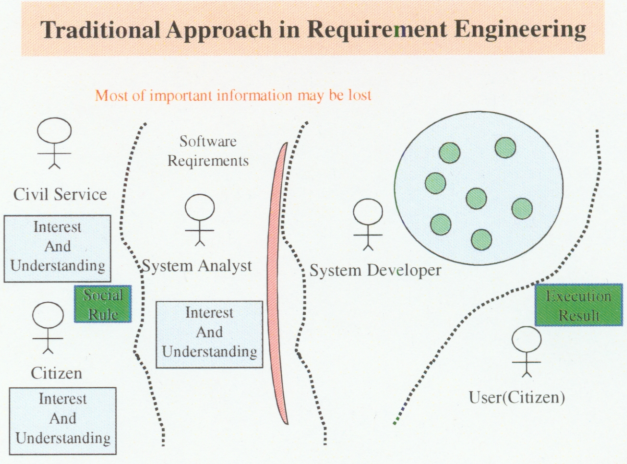

Fig.4 Problems in traditional approach

In traditional requirement engineering approach, the result of requirement elicitation is tranformed into functional/non-functional requirements of the system. Most of important information that supports software accountability may be lost. **Goal-Oriented Requirement Engineering** approach [4,5] has been studied recently, and it has a strong relationship to our aim and approach to define software accountability ‡ .

---

‡ Mr. Shuichiro Yamamoto of NTT data taught me the strong relationship between GORE and the consideration I did so far

In the Goal-Oriented Requirement Engineering (GORE) approach, non-functional requirements such as Ease-of Maintenance and Goodness of Usability are defined first as goals. And then the goals are unfolded by an AND-OR tree with defining the sub goals. Functional requirement is allocated at the leaf of the tree.

One of the features of this approach is in "Soft Goal". Different from the definition of goal in Artificial Intelligence research area, they have different definition on satisfiability of a sub goal.

- A sub goal is satisfied if there are enough affirmative evidences and there are few negative evidences.
- A sub goal is not satisfied if there are enough negative evidences and there are few affirmative evidences.

Adopting GORE enables us to represent the semantics understood by stakeholders and their relations, with forming the layer structure as shown in Fig.5.

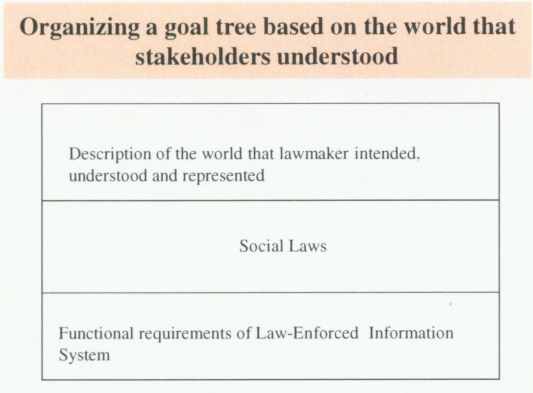

Fig. 5 Organization of a goal tree

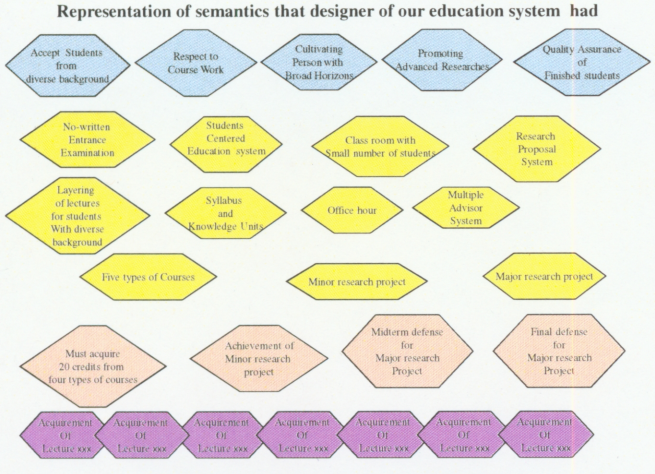

Fig.6 Layers of goals / sub goals considered by the stakeholders of our education system

We show the examples of elements of each layer in Fig.6 based on the idea shown in Fig.5,. We took an example from the curriculum of our Institute, JAIST. The goals and sub goals in Fig.6 are considered and decided by the stakeholders who were related to the establishment of JAIST.

Goals and sub goals shown in Fig.6 is useful for

- Controlling the evolution of our curriculum, an example of a social rule
- Answering the questions that ask us the origin of our rule establishment

3.2 Consideration from legal theory standpoint

According to the legal theory by Torstein Eckoff [6], **Legal System** consists of **Norms** and **Acts** and there are various kinds of relations among them. In this section, we summarize his theory with citing the related part of [6].

Norm is a generic term for rule, principle, regulation, standard, pattern, guide, criterion and classified into Direction , Qualification , Authorization. Linguistic representation of normative expression is called **Normative Statement**.

- **Direction** is a generic term for order, demand, entreaty, advice, warning, and promise. It directly represents intention to make someone's mind changed.
- **Qualification** is a concept that corresponds to a deductive rule in mathematics. It shows what phenomenon belongs to which category.
- **Authorization** is an official permission to do something. It confers power to someone to give rights, to qualify, and to command. Authorization plays a central role in legal system. Examples are: enactment of a basic law on legislative power and judicature; law that confer the power of decision to the Civil Service.

Norm that includes direction or negative direction as linguistic elements is called **Duty Norm.** Duty norm has four sub groups. Those are order, prohibition, permission, exemption. We are obliged to start the ordered act. We are obliged to stop the prohibited act. Permitted act includes ordered act and discretion (neither ordered nor prohibited) act. Exempted act includes prohibited act and discretion (neither ordered nor prohibited) act.

Subjects of Duty Norm are categorized into two.

One of them is individual or persons who is/are imposed the duty. Another is a group that is directed the

duty. The latter position is often characterized by the word "right".

Direction can be understood as either Duty Norm or Qualification depending on the standpoint we take.

There are relations between Norms, and between Norm and Act. Relation between Norms has two types, static relation and dynamic relation.

Static relation is a relation that is not affected by the change in the legal system.

- **Coupling relation** When two or more norms are composed into a whole perfect normative statement, those norms are connected by the coupling relations. Some law has a definition of something. And the definition appears in many other laws. They have coupling relation. For an example, many laws refer the law that defines the term "relative".
- **More-than-one-semantics** Some expression carries more than one statement. Suppose some law expresses that the authorities concerns can direct something. This expression means not only "the authority has power" but "the decision whether to exercise the power or not is left to the authority".
- **Logical relation** logical equivalence, inclusion, contradiction between norms

Dynamic relation is the relation between steps in the flow of real or imaginary act

- **Causal relation** A relation between the first step where some event occurs (cause) and the second step where the other event occurs (effect). The cause causes the effect. Causal relation can link only facts (status, act, event)
- **Normative relation** A relation between facts. But it is not a causal relation but a relation based on norm. The fact "Mr. A stole" and the fact "Mr. A received a sentence of a two-year prison" are connected by a norm. Normative relation between facts sometime means that there is Causal relation too.
- **Operational relation** If norm applied at some step decide the norm to be applied at the next step, there is a operational relation between two norms.

Act is an element of a legal system too. The major activities performed by some legal organization are: prepare for the various kinds of decisions; decide; and explaining those decisions. Legal system accepts two inputs. Those are "support the realization of legal claims" and "change the legal system itself". There are two dynamic processes, Application of Law and Enactment of Law.

- **Application of Law** The characteristic of this process is that norms and evaluation of norms form the evidence of decisions.
- **Enactment of Law** means creation, change, and abolish of norms.

Deliberation Process is important in both enacting and applying laws. Deliberation process inputs the problem and data, and outputs Position and Explaining.

- **Position** is a statement to something, such as "what should it be", "What should not it be" and "what should be done".
- **Deliberation** is a psychological process that brings someone into some Position.
- **Explaining** explains, expands, and supplements some Position.

We can type the social rules by using legal theory. And it enables us to give a basis for realizing the method to extract properly related subset of social rules for a question.

3.3 On realizing the software accountability function

We can define and realize the software accountability function of LEIS by applying the results discussed in section 3.1 and section 3.2

- We can organize the semantics of the world understood by stakeholders as a goal-oriented tree.
- It is possible to define the world intended, understood, and represented by lawmakers in the first layer of Fig. 5, using the input and output of dynamic processes, application of law and enactment of law, described in section 3.2. We can define goals based on: relations between "problems and data" and "position and explaining" or "explaining the position" itself.
- For examples, the goals of our education system and major sub goals shown in Fig.6: Accept students from diverse areas; Respect to course work; Cultivating person with broad horizons; Promoting advanced researches; Quality Assurance of finished students; Non-written examination; Office hour; Major and Minor research projects; and so on; were considered and decided by the top level governmental committee. Some of sub goals were considered and decided by the internal committee of our institute. Those are, for examples, Research proposal system; Syllabus and related knowledge

units; and so on.

- Finally the rule for completion was established to give directions to student as a curriculum. They were: Must acquire 20 credits from four types of courses; Achievement of major and minor research projects; Pass for midterm and final defenses and so on along a time schedule for completion.
- The information existing in upper layers are essential and important information for our institute to evaluate and revise the curriculum. That is to say, information related to Deliberation, Position and Explaining has a strong relation with the stakeholders' interest, and necessary information source to generate answer for the questions.
- So far, we discuss how to organize the layer structure, but it is important to discuss how to set the relationships between rules in specific layer, especially for the second layer in Fig.5.
- There is a strong relationship between this topic and consideration in section 3.2 in structuring the information at some layer. That means we can answer the question like "Why can I not submit my research proposal?" by using the causal relationships and/or normative relationships set among rules.

We define the **Accountability tree** here that organizes the elements in Fig.6 by a goal-oriented tree in GORE and each leaf of the accountability tree is typed by the legal theory. We show an example of an accountability tree in Fig.7.

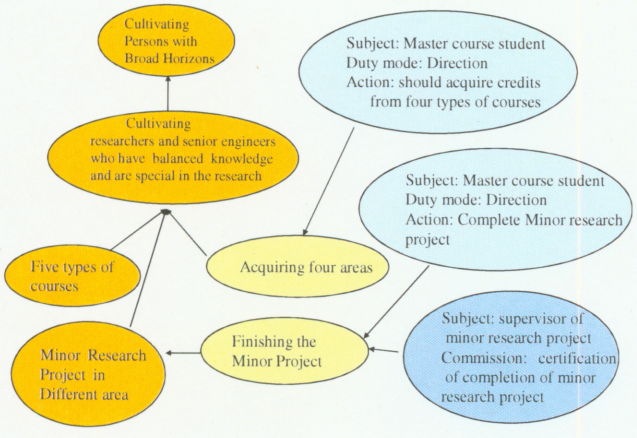

Fig.7 An example of the accountability tree

4. Consideration on realizing the software accountability function

If some LEIS itself can answer the question from various types of stakeholders, we say the LEIS has accountability functions. In this section, we first consider where the necessary information exists in the development process of LEIS. We show the supposed process in Fig.8.

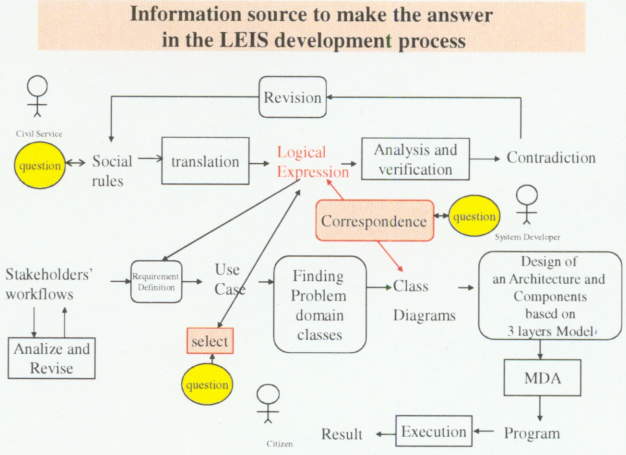

Fig.8 Information source to produce the answer for three types of questions in the LEIS development process

Outline of LEIS development process is shown below.

- Social rules written in natural language are automatically translated into logical expressions.
- The logical expression is analyzed by using legal inference to detect contradictions. Resolution is done manually. We call this cycle **legal debugging**.
- The logical expression is also an input to LEIS design. We consider two ways as follows;
  - Generate a class diagram from logical expressions.
  - Define stakeholders' workflow to apply a social rule first. Then define a use case model using both workflow and the logical expression. Finally problem-domain classes are found following the use-case driven software development approach.
- Generate a system based on MDA approach, adopting Three-Tiered model.

In the process mentioned above, Information related to software accountability is obtained from the points depicted in Fig.8.

- **Logical Expression** is primary information for type 1 software accountability.
- **Correspondence** between the logical expression and the class diagram is primary information for type 2 software accountability.
- **Logical expression, Correspondence and execution**

history of the system is primary information for type 3 software accountability.

Software accountability functions of LEIS are realized as follows;

- **Type 1 software accountability**

  Realized as one of the functions of legal debugger.

Type 2 software accountability

Providing the Database and a query system

- **Type 3 software accountability**

  We provide the software **accountability module** as shown in Fig.9.

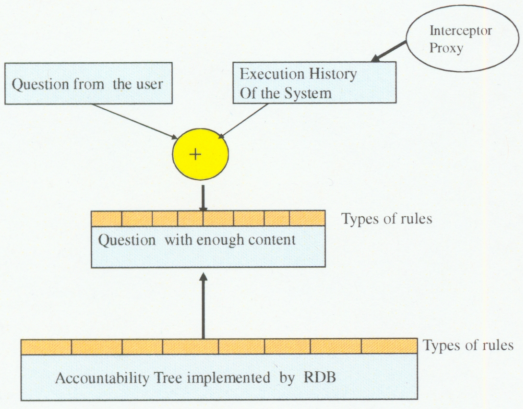

Fig.9 Internal Structure of the Accountability Module

We provide the interceptor proxy introduced in the next section that can record the execution history of the LEIS. We generate the question with enough content by combining stakeholders' question with the execution history. We extract the corresponding social rule and related ones by retrieving the accountability tree using the vector space method. And generate the proper answer to the question.

5. Realization of Software Accountability Functions in the Course Management System

In this section, we show the result of the feasibility study to realize software accountability functions for type 3 in the course management system.

It is common that LEIS is a web-based system which is modeled as a three-tier architecture [9] including the user system interface tier, the process management tier, and the database management tier. We proposed the reference architecture to realize software accountability function based on the three-tier architecture [8] (Fig.10).

A feature of the reference architecture is in the extensible mechanism which enables us to attach an

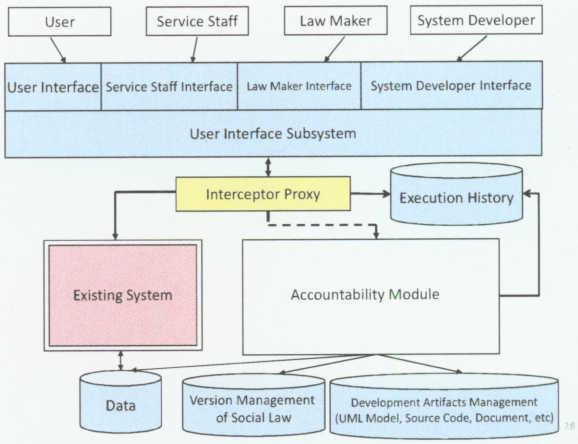

Fig.10 The reference architecture to realize the software accountability function based on the three-tier architecture

accountability module to an existing system, by placing the interceptor proxy between the user system interface tier and the process management tier. Accountability functions are realized by using the two databases. Those are the execution history of the system and the social laws with a version management mechanism.

We have done a case study for a course management system.

- Firstly, we have developed the course management system using a use case driven development approach. The system is a web-based application of a client-server style using the Java EE (Java Platform, Enterprise Edition 5) platform.
- Then, we have attached an accountability module to it based on the reference architecture.

In this section, we show the mapping between the reference architecture and Java EE platform, and the realization method for JBoss Seam web application framework [11], which is one of the implementation of Java EE.

We describe an overview of the course management system we have developed and the system structure of it in section 5.1. In section 5.2, we discuss a realization method of accountability functions in the system. In section 5.3, we show an execution example of accountability functions in the system.

5.1 The Course Management System

We have developed the course management system using COMET [12], use case driven object-oriented development methodology. The system supports us to apply the registration rules of our institute. We show the overview of the course management system and the structure of the system using the development artifacts,.

5.1.1 Use Case Diagram and Use Case Description

We started from capturing the needs by sending questionnaires to the students of our school. They are the expected users of our system. Analyzing the questionnaires, we obtained the functional requirements of the system.

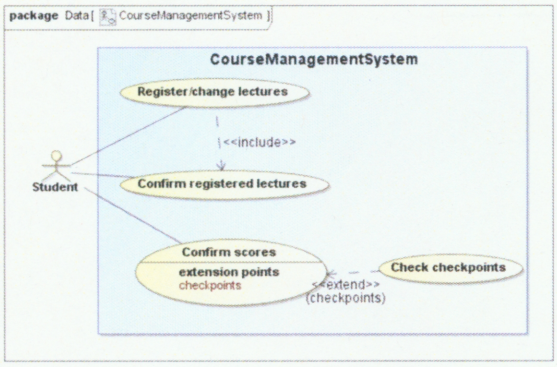

Fig. 11 Use case diagram of the course management system

Part of the use case model we defined is shown in Fig 11. Although we need to model all of the stakeholders such as students, staff of the institute, rule makers, and system developers as actors, we model only students in the master's program as an actor in this case study.

A student can register or change the courses that he/she wants to take, and then confirm the contents of the registration. "Confirm scores" use case shows a list of the courses the student has been acquired. The list consists of a course name, the units of a course, and a score of a course.

"Check checkpoints" use case (Fig. 12) is executed when a student requests to check the checkpoints in "Confirm scores" use case.

In order to complete the master's program of our institute, student should clear all of the check points. There are four checkpoints defined in the registration rules: "the requirement to start a minor project", "the requirement to submit a research proposal", "the requirement to take job-recommendations", and "the requirement to complete the master's program". Students cannot pass each of the checkpoints unless he/she satisfies the requirement of the checkpoint. "Check checkpoints" use case gives a student the checked result using him/her scores.

- **Use case name:** Check checkpoints
- **Actor:** Student
- **Precondition:** Student is logged in. DB of registration rules is available.
- **Description:**
  1. A student selects a checkpoint from a list of the checkpoints.
  2. The system queries student's score from a DB of student scores.
  3. For all the conditions of the checkpoint
     1. The system checks whether student's score satisfies the condition or not.
     2. The result is put into the check_vector.
  4. The system outputs the check_vector and the result of checking the checkpoint.
- **Postcondition:** Student obtains information of the vector and the result.

Definitions of terms:

- check_vector : := (check_result check_item check_difference)*
- check_result : := true | false
- check_item : := number of course types | number of units for each course types | number of total units | number of units for each course layers | units of minor research project
- check_differnce ::= the value of check_item specified by the rule minus the value of check_item of student's scores
- checkpoint : := beginning minor project | submitting research proposal | accepting job recommendations | graduating master's program

Fig. 12 Description of "Check checkpoints" use case

5.1.2 Page Transition Diagram

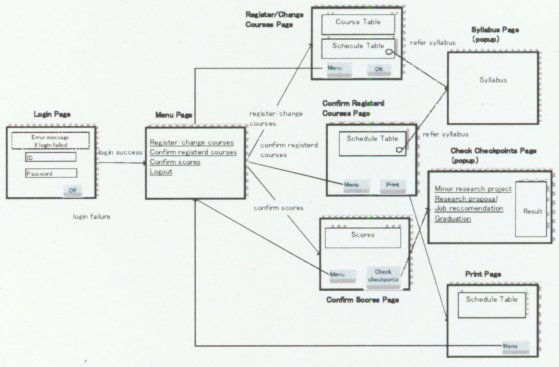

Fig. 13 Page Transition Diagram

The design of page transitions and page layouts is necessary for web applications. The course management system is a web application, so we designed the page transition diagram (Fig. 13) after defining the use case models.

Each of the use cases in the use case model corresponds to each item in Menu page. All pages except Login page have a Menu button which navigates a logged-in student to Menu page at anytime.

5.1.3 Platforms and Web Application Frameworks

There are several web application frameworks we can use now. In this development, we selected the Java EE platform and JBoss Seam as a web application framework. JBoss Seam unifies and integrates JSF (JavaServer Faces) for a presentation tier and EJB (Enterprise JavaBeans) 3.0 as a distributed component technology. It also provides multiple stateful contexts and declarative state management.

5.1.4 Class Diagram

The class diagram is shown in Fig.14. The local interfaces of EJB Session Bean are omitted.

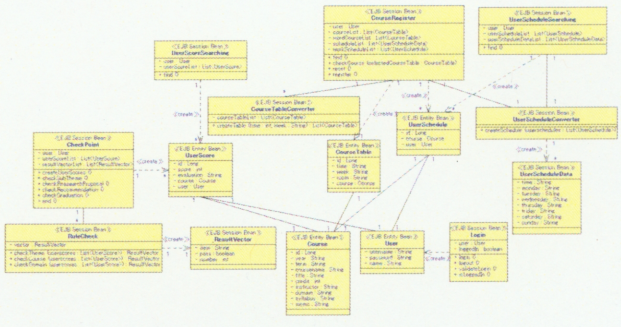

Fig. 14 Class Diagram of the Course Management System

In Fig.14, <<EJB Entity Bean>> stereotype means the O/R (Object / Relational) mapped classes, which properties are persisted into relational databases. <<EJB Session Bean>> stereotype means the classes implementing business logics which methods are called when events in the presentation tier like page transition occur. Some of properties in the stereotyped classes <<EJB Session Bean>> are implemented by using DI (Dependency Injection) mechanism provided by JBoss Seam.

5.2 Realization of Accountability Functions in the Course Management System

We discuss a method to attach the type 3 accountability module mentioned in the previous section to the course management system.

5.2.1 Accountability Functions and Realization Approach

The reference architecture (Fig. 10) shows that accountability functions are realized by using a database of execution history of the system and social rules database. We consider two methods to get the history.

Accountability functions realized is different from each other because the information can be different for each method.

The first way is to get the history only from an interceptor proxy inserted between the user system interface tier and the process management tier. The interceptor proxy can get the data on method calling between two tiers, and store the data into the database. Using this way, accountability functions can be added to the existing system with no modification to the system, but the history executed in the process management tier in the system cannot be collected. Therefore, the only data about method calling (method name, arguments, return value) from the user system interface tier to the process management tier can be used as the history. The accountability functions realized is limited.

The second way is to get the history from the process management tier in the existing system. in addition to an interceptor proxy. For an example, if the existing system is implemented by Java, applying AOP (Aspect-Oriented Programming) technology can get the history from the process management tier without modification of code. It is possible to get the history from the Java VM using JDI (Java Debug Interface), which is provided mainly for debuggers. Using this way, the history such as execution paths of business logics, change history of observed variables etc can be collected. It, however, requires us to understand the code of the existing system. Therefore, richer accountability functions could be realized than the first one.

This time, we adopted the first method. The accountability functions of the course management system are defined below.

Clicking one of the items of checkpoints, the system checks whether the student's scores satisfy the conditions of the checkpoint or not, and shows the result of checking on Check Checkpoints Page in Fig. 13. If a student does not satisfy the result shown by the system and ask a question to the system, the system explains the following reasons why it made such decisions or calculation using the accountability functions.

- The origin of the rule establishment
- Registration rules related to the checkpoint
- The student's scores
- The conclusions led from applying the rules to the situations (the scores)

5.2.2 Mapping between the Reference Architecture and Java EE architecture

The reference architecture (Fig. 10) is based on tree tier architecture so that it can be mapped to Java EE architecture because Java EE architecture is also based on tree tier architecture. Fig. 15 shows the Java EE architecture to realize accountability.

The Java EE architecture showed in Fig. 15 does not need an interceptor proxy because Java EE includes an interceptor mechanism in its specification. The architecture only needs interceptor components collecting execution history. In Java EE, an call of accountability function is mapped to an call of the method of session beans of the accountability module in the process management tier.

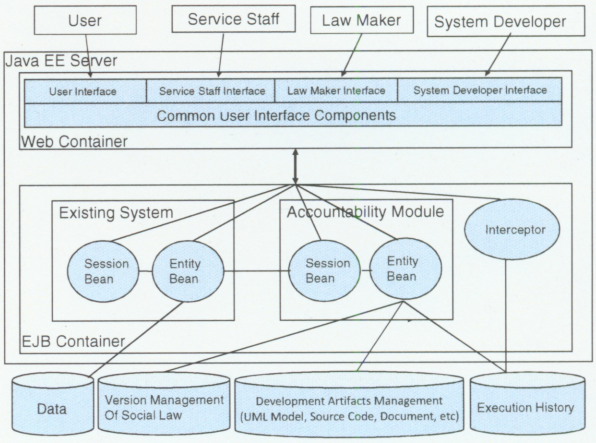

Fig. 15 Java EE architecture to realize accountability

5.2.3 Realization Method of Accountability Functions Using JBoss Seam

In this section, we discuss how the accountability functions are realized in the course management system using JBoss Seam, which is an implementation of a web application on Java EE platform.

User Interfaces

A part of modifications for the user interface is necessary to accept requests of calling accountability functions and to present the explanations generated by the accountability functions. We adopted the buttons form of user interfaces for accountability functions, but we think we need more discussions about the user interface design. The buttons are placed on the page on which the system displays some results of executions, and if a user clicks the button, the corresponding accountability function is invoked. The explanation for the question is presented on a popped up window.

In the course registration system, "Why?" buttons that invoke the accountability functions are added to the part of presenting the result on Checking checkpoints page. To implement this, xhtml files defining the view part of

presenting the result on the page are modified to a few lines only. The lines include the name of methods of session beans which are called by clicking the buttons. When a student clicks a "Why?" button, a popped up window is open, then the explanation about the reason why the student satisfies the checkpoint or not is displayed on the window.

Interceptors

JBoss Seam supports interceptor mechanisms which enables the system to collect an execution history.

This is done by the following steps. Firstly, the interceptees which are components called from the user system interface tier are specified in the ejb-jar.xml file. Then, defining interceptor classes shown in Fig. 16, the pre processing and the post processing log the invoking component name, the method name, the argument values of the method, and the return values of the method into the execution history database.

```
@Interceptor(type=InterceptorType.CLIENT)
public class LoggingInterceptor {
    @AroundInvoke
    public Object logging
      (InvocationContext invocation)
      throws Exception {
       // PRE PROCESSING
       Object result = invocation.proceed();
       // POST PROCESSING
      }
}
```

Fig. 16 Coding Example of Interceptors

Accountability Module

The accountability module providing accountability functions is also implemented using EJB components on the Java EE platform. We designed each of "Why?" buttons which corresponds to a session bean, implementing the logic to explain the reason in the accountability module.

For example, when a student has a question about the result of checking the requirement of "Starting a minor project" on "Check checkpoints" page, clicking the "Why?" button next to the link of the checkpoint items, the method of the session bean which implements the explanation logics about the requirement in the accountability module is invoked.

This method implements the logics described in the section 5.2.1 which explain the reason why the decisions or calculations are made. The first and second of the explanation logics use the social rules database, and the third uses students' scores database which is one of the database of the existing system. The fourth composes the satisfactory explanation sentences using the data retrieving from the databases, and returns the explanation to the user system interface tier.

Social Rule Database

We realized the accountability tree defined in section 3.3 as a relational database. We defined the accountability tree for the rules-for-completion of our institute first. And then design the database scheme as shown in Fig.17 based on the accountability tree we defined [13].

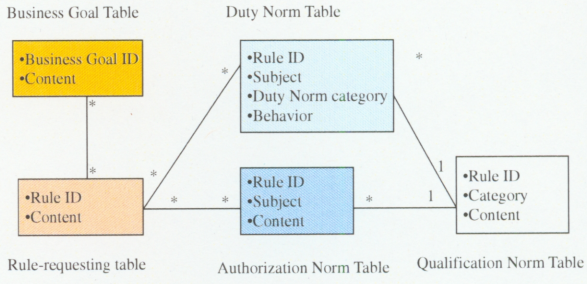

Fig.17 Implementation of an accountability tree by relational database scheme

We explain the tables in Fig.17 below.

- Business Goal Table
  This table contains the goals and sub goals (nodes in Fig.7) of the accountability tree.
- Rule Table
  This table contains the leaf of the accountability tree in Fig.7. Each leaf contains the directions to student, for examples: Must acquire 20 credits from four types of courses; Achievement of major and minor research projects; Pass for midterm and final defenses.
- Duty Norm Table
  This table contains rules that belong to the Duty Norm category. Attributes of this table are subject, duty category and behavior.
- Qualification Norm Table
  This table contains rules that belong to the Qualification Norm category. Attributes of this table are category name and content of category. For an example, category name: Supervisor of minor research project; content of category: professor or associate professor assigned by Dean at the beginning of the research.
- Authorization Norm Table
  This table contains rules that belong to Authorization Norm. Attributes are subject and content of authority.

The accountability module composes the "question with enough content" by adding the data collected by the interceptor proxy to the question that student wrote freely(see Fig.9). The accountability module computed the similarity between the question and the content of database. Then the module retrieved the most related rules with the question [14]. Most of question and rules do not include the subject of behavior. The module supplements it.

The conclusion of the experiment is that it is better to calculate the similarity of "the subject of behavior" and "content of behavior" independently.

Providing the version control mechanism for social

rule database to support the evolution of social rules is a future issue to be studied.

5.3 An Example of Executing the Accountability Functions

The "Check checkpoints" page in the page transition diagram (Fig.13) implements "Check checkpoints" use case. When a student clicks an item of the four checkpoints, the result of the checks is presented. The student can ask the system about the result by clicking the corresponding "Why?" button, invoking the accountability functions. Fig. 18 shows an example of executing the accountability function.

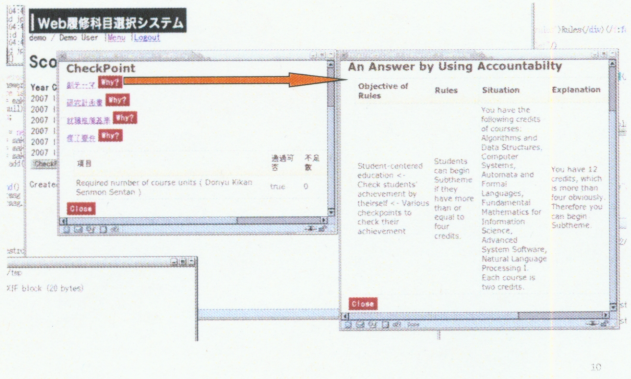

Fig. 18 Example of Executing Accountability Functions

In Fig.18, the "CheckPoint" window displays the result of checking the "Starting a minor project" checkpoint. A student can ask the system about the result by clicking the "Why?" button next to the link of "Starting a minor project."

When the student clicks this "Why?" button, the explanation of the execution result as a table format is presented in a popped up window. The table includes four columns: the origin and purpose of registration rules related to starting a minor project, the rules, student's situations (student's scores), and the explanation statements. The rules and their origin and purpose are retrieved from the social rule database. The student's scores are retrieved from the execution history database. In this way, the accountability module produces satisfactory explanations using these data from the databases.

It is necessary for considering and improving the explanation format rather than the table format though the system we developed is still a prototype system. The student knows and understands the reason why he/she can start a minor project. This accountability function helps students satisfy the result of the checks done by the system.

6. Conclusion

In this paper, we defined the software accountability and discussed the architecture to combine the software accountability module with LEIS.

We showed the organization of the software accountability module that is structured by GORE and is typed by the legal theory.

We also showed the case study by developing the course management system that proves the feasibility of our approach. We adopted Java EE architecture and JBoss Seam to build the system.

We are now studying the elaboration of our theory at the following points.

- Refinement of attributes of an accountability tree to support the evolution of the social rule.
- Finding heuristics related to the vector space method to improve the performance.
- Refinement of data collected by an interceptor proxy and improving its performance.

Course management system with software accountability itself is being improved for real use.

Acknowledgement

We thank to Professor Takuya Katayama for his useful suggestion to promote this research. We also give many thanks to Professor Mitsuru Ikeda, Associate Professor Masato Suzuki, Assistant Professor Kazuhiro Fujieda, and Assistant Professor Satoshi Hattori for their useful discussion.

References

[1] Takuya Katayama, "Verifiable and Evolvable e-Society – Realization of Trustworthy e-Society by Computer Science", The Current status of the Art of the 21st COE Programs in the Information Science Field(2), Journal of Information Processing Society Japan, Vol.46 No.5, pp.515-521, 2005. (in Japanese)

[2] Longman, Advanced American Dictionary.

[3] Koichiro Ochimizu, "Defining Software Accoun tability" The Institute of Electronics, Information and Communication, IEICE Technical Report, SS2006-33, pp.49-54, 2006. (in Japanese)

[4] John Mylopoulos, Lawrence Chung, and Eric Yu, " From Object-Oriented to Goal-Oriented Requirements Analysis", CACM, Vol.42 No.1, January 1999.

[5] Shuichiro Yamamoto, "Goal-Oriented Requirement Management Techniques of Information System", Software Research Center, 2006. (in Japanese)

[6] Torstein Eckoff and Nils Kristian Sundby, "RechtsSysteme", 1988. Japanese Translation by Hiromi Tuzuki, Kazuyosi Nozaki, Takahiro Hattori, Itaru Matumura, Mineruba Shobou, 1997.

[7] Ryo Hayasaka, Hirotoshi Akiyama, Hayato Sugimori, Sintaro Kitayama, Masato Suzuki, Koichiro Ochimizu, "The Design and Implementation of Software Accountability Functions for Course Registration System", The Institute of Electronics, Information and Communication, IEICE Technical Report, SS2007-14,

pp.29-34, 2007. (in Japanese)
[8] Ryo Hayasaka, Masakazu Hori, Kazuhiro Fujieda, Koichiro Ochimizu, "Applying the Software Architecture with Accountability and Ease-of- Evolution to the Three-Tier Model", Information Processing Society of Japan, IPSJ SIG Technical Report, 2005-SE-150, pp.1-8, 2005. (in Japanese)
[9] G.Schussel: "Client/server: Past, present and future" (1995) online.
[10] D.C.Schmitdt, M.Stal, H.Rohnert and F.Bushmann : "Pattern-Oriented Software Architecture Volume2 – Networked and Concurrent Objects", John Wiley and Sons(2000).
[11] JBoss.org:"JBossSeam", http://labs.jboss.com/jbossseam/
[12] H.Gomma: "Designing Concurrent, Distributed, and Real-time Application with UML", Addison-Wesley (2000).
[13] Hayato Sugimori, Koichiro Ochimizu, " Report on Realizing Software Accoountability by using GORA and Legal theory, JAIST Research Report, IS-RR-2007-005, 2007. (in Japanese)
[14] Shintaro Kitayama, Koichiro Ochimizu, "Primary Experiment on retrieving social rules using the Vector Space Method", JAIST Report, 2007. (in Japanese)