Institute of Electronics, Information, and Communication Engineers

PAPER Special Section on Knowledge-Based Software Engineering

A Method to Develop Feasible Requirements for Java Mobile Code Application

Haruhiko KAIYA[†a], **Kouta SASAKI**[†], *Nonmembers, and* **Kenji KAIJIRI**[†], *Member*

SUMMARY We propose a method for analyzing trade-off between an environment where a Java mobile code application is running and requirements for the application. In particular, we focus on the security-related problems that originate in low-level security policy of the code-centric style of the access control in Java runtime. As the result of this method, we get feasible requirements with respect to security issues of mobile codes. This method will help requirements analysts to compromise the differences between customers' goals and realizable solutions. Customers will agree to the results of the analysis by this method because they can clearly trace the reasons why some goals are achieved but others are not. We can clarify which functions can be performed under the environment systematically. We also clarify which functions in mobile codes are needed so as to meet the goals of users by goal oriented requirements analysis(GORA). By comparing functions derived from the environment and functions from the goals, we can find conflicts between the environments and the goals, and also find vagueness of the requirements. By resolving the conflicts and by clarifying the vagueness, we can develop bases for the requirements specification.

key words: *goal oriented requirements analysis, anti-requirements, security policy, access control, Java mobile codes*

1. Introduction

Java applications can use several mobile codes provided by different code providers at the same time. We can construct and execute various kinds of applications easily and efficiently with the help of these codes. In a machine where such an application is executed, there are valuable resources, such as files and network connections, that should be protected from malicious and/or inadequate access by mobile codes. In the case of Java mobile codes, all functions accessing such valuable resources are revoked by default. Granted functions for some resources are defined in a description of security policies, so that functions required or permitted by users are enabled.

However, it is not so easy to know whether such description of security policies consistently meets the *goals* of the application users. For example, most users can not easily identify inadequate functions until inadequate results happen. However, some inadequate functions can be granted by the description unconsciously, and goals such as 'something should not happen' are not achieved. Even when we develop an application without mobile codes, it is not so easy to know such negative requirements. Fortunately, inadequate results do not frequently happen in an application without mobile codes because most codes are tailored only for the application. However, inadequate results happen easily in an application with mobile codes because mobile codes are tailored for many applications. In addition, using mobile codes is almost the same that mobile codes' developers access user's own machine directly.

Even if we can identify all inadequate functions by a mobile code and we can write security policies so as to avoid such functions, another kind of problem happens. In Java security system, grant rules are applied not to each mobile code but to each location where mobile codes are placed. Consequently, functions to valuable resources are granted to all codes placed in the same location. In addition, we can not easily change the deployment of such mobile codes because mobile codes are provided by other companies and/or organizations. As a result, we can not sometimes avoid inadequate functions intrinsically, therefore we should abandon some goals so as to develop an application under such *an environment* including the deployment and features of mobile codes and security policies.

As a result of above discussion, it is realistic to compromise the differences between an environment and goals, and to identify what kinds of things are compromised and why they are compromised. During such compromising process, we can get feasible requirements for mobile code applications, and also have a chance to clarify vagueness of requirements. In this paper, we will introduce a method for such purpose. In our method, we explore the possibility to refine goals so as to meet an environment as follows.

1. Clarify the functions required by users' goals and the functions enabled under an environment.
2. Identify the differences between these two kinds of functions.
3. Resolve conflicts between them by modifying the environment and/or by abandoning some parts of goals.
4. Clarify vagueness of goals by specifying functions derived from the goals.

We use goal oriented method [1] for analyzing requirements. We use our method to generate and check security policies under a deployment of mobile codes [2] for analyzing the functions derived from an environment for mobile codes.

In Sect. 2, we show the overview of the method and introduce a tabular form for representing functions. In Sect. 3, we discuss which functions in mobile codes can be executed under an environment. In Sect. 4, we discuss which functions are required by users. In Sect. 5, we show how to

Manuscript received June 29, 2003.
Manuscript revised October 8, 2003.

[†]The authors are with the Faculty of Engineering, Shinshu University, Nagano-shi, 380–8553 Japan.
a) E-mail: kaiya@cs.shinshu-u.ac.jp

NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers

identify and compromise the differences between the available functions under the environment and the users' requirements. Examples using our method are shown in Sect. 6. In Sect. 7, we refer the related works to clarify advantages and limitations of our work. Finally, we summarize our results and show the future direction.

2. Overview of the Method

In our method, we write several kinds of descriptions related to our requirements analysis. By modifying these descriptions stepwise during the analysis, we achieve trade-off between environments and goals. In Fig. 1, we show descriptions that are used and written during an analysis, and their relationships. In Fig. 2, we also show descriptions and relationships after the analysis. Using these figures, we will outline the procedure how to analyze the trade-off between environments and goals.

The main inputs of our method are *an environment description* and *a goal hierarchy*. The environment description represents the following contents; functions and features of each mobile code, their deployment over the network and security policies for a site where intended application will be executed. The goal hierarchy represents what application users want.

We derive *enabled functions* and *required functions* from an environment description and a goal hierarchy respectively. Enable functions represent the abilities accessing valuable resources, and required functions represent the requirements accessing valuable resources. In our method, we identify and compromise the differences between enabled and required functions by comparing them directly. Therefore, we describe enabled and required functions in the same tabular form. We call such tabular forms as *an enabled functions table* and *a required functions table* respectively. By using these tables, we can easily and systematically find conflicts and vagueness.

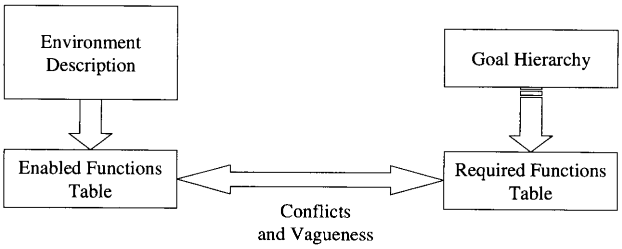


Fig. 1 Descriptions during an analysis.

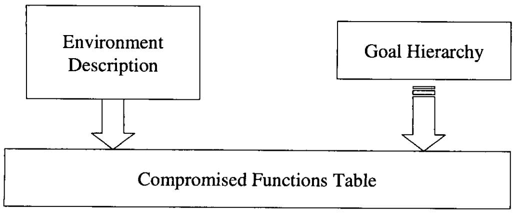


Fig. 2 Descriptions after the analysis.

Because terms in enabled functions are normally those in implementation level, such terms are not fit for representing requirements in general. In our method, we encourage analysts to decompose and refine the goals in a goal hierarchy until terms in the goals can fit for terms in enabled functions.

The main output of our method is *a compromised functions table* in Fig. 2, that becomes bases of software requirements specification. Compromised functions in the table can be performed under the given environment, but are not always consistent with a goal hierarchy for the application.

In our method, an environment description and a goal hierarchy are modified so that required functions become consistent with enabled functions. As a result, required and enabled functions tables become the same after the analysis. The name of 'compromised functions' is only an alias of the name of required or enabled functions after the analysis. Therefore, the notation of compromised functions table is also the same as the notation of required and enabled functions tables. Our method guarantees compromised functions to be feasible under the given environment, but the functions are not always consistent with the goal hierarchy.

The goal hierarchy is modified and/or extended when implicit and/or unidentified goals are found by using this method. The environment description is also modified when the environment can be modified in fact and such modification meets required functions derived from the goal hierarchy. As a result, the goals become clearer, and the environment becomes fit to user goals if possible.

2.1 Tabular Form for Functions

We represent both enabled and required functions tables like Table 1. The left hand column of the table shows valuable resources such as data and/or objects that will be accessed by functions. The top row of the table shows mobile codes that will execute functions.

Values in cells of the table show the functions. For example in Table 1, an application cannot access any valuable resources using mobile code Read.class. On the other hand, it can read and write files under /home/ using Write.class. It can also make any network connections using Net.class. Note that `r` and `w` are abbreviation for read and write.

Enabled functions are completely determined in general. On the other hand, required functions are intrinsically incomplete because they are derived from goals of application users. Therefore, we attach the following postfixes to each function in a table. The word '`function`' represents some function, e.g. `r`, `w` or `connect`.

- `function+`: The function is enabled or required.

Table 1 An example of enabled functions table.

	Read.class	Write.class	Net.class
/home/	-	r+ w+	-
any ports	-	-	connect+

Institute of Electronics, Information, and Communication Engineers

- `function-`: The function is not enabled, or disablement of the function is required.
- `function*`: We don't care whether the function is required or not.
- `function?`: We don't have decided whether the function is required or not.

We frequently use the abbreviation ‘+’ in a cell which means all possible functions are enabled or required. We also use the abbreviation ‘-’, ‘*’ and ‘?’ in the same way. We sometimes abbreviate ‘function+’ as ‘function’. All of the not specified part is filled with ‘-’ by default. If there is no confusion, we may abbreviate the name of valuable resources and/or mobile codes. For example, we may abbreviate ‘/home/’ to ‘home’, and ‘Write.class’ to ‘Write’.

3. Which Functions in Mobile Codes can be Executed?

In this section, we will show how to derive enabled functions from an environment description.

3.1 Access Control Mechanism in Java2

As mentioned in a survey article [3], the expression ‘mobile code’ has various different meanings. In this paper, we only focus on quite simple applications using mobile codes as shown in Fig. 3. An application is executed on a machine, and it downloads and uses several mobile codes from several different sites. We may locate a part of mobile codes in the machine. We call a map from each code to a location where the code is placed before download, as a *deployment* in this paper.

Figure 4 shows an example of policy description for Java [4]. As mentioned in the first section, all functions accessing valuable resources are revoked by default, and granted permissions are given to each location. The policies in Fig. 4 grant the following permissions to applications using mobile codes.

- The functions reading and writing files in /home/* can be executed if that functions are embedded in codes in http://www.a.b/otherdir/class/.
- The functions making network connections using any ports can be executed if that functions are embedded in codes located in ftp://ftp.a.b/somdir/net/.

When the policies in Fig. 4 are applied to an application running in a machine in Fig. 3, the application cannot access any valuable resources in the machine using mobile code Read.class in http://www.a.b/somdir/java/. On the other hand, it can read and write files under /home/ using Write.class in http://www.a.b/otherdir/class/. It can also make network connections using Net.class in ftp://ftp.a.b/somdir/net/.

In general, granted permissions are defined for each location. Locations are represented in an URI form, and permissions are represented in the following form.

permission_class_name target_name action_list

There are several standard permission_classes, such as FilePermission and SocketPermission, in Java2 security system, and we can append new non-standard permissions. The targets and the actions are defined in each permission_classes. The targets show valuable resources for a permission_class, and actions show kinds of functions applied to the targets.

3.2 Enabled Functions and Their Derivation Algorithm

Security policies specify the enabled functions that can access valuable resources and that can be executed in a mobile code application under an environment. We will show an algorithm to derive enabled functions.

1. Enumerate the functions that access valuable resources and that can be called from each code:
   We can know such functions embedded in the standard class library of Java beforehand from their documents. We can find such functions in the non-standard libraries and handmade codes by analyzing them. Even when source codes are not available, we can do it with the help of Jode [5].
   In both cases, we can not strictly enumerate such functions because classes and methods are dynamically

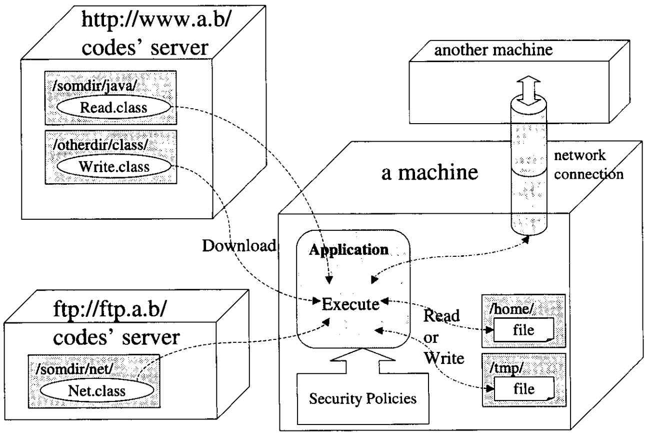


Fig. 3 Mobile code application and its environment.

```
grand codeBase
  "http://www.a.b/otherdir/class/" {
    permission java.io.FilePermission
         "/home/*", "read, write";
}
grand codeBase
  "ftp://ftp.a.b/somdir/net/" {
    permission java.io.SocketPermisson
         "*", "connect";
}
```

Fig. 4 An example of policy description.

NII-Electronic Library Service

Institute of Electronics, Information, and Communication Engineers

bound in Java runtime. As a result, some functions to access valuable resources are always called from the code, but others are not always called. In this paper, we focus on all functions which have the possibility to be called, because we want to explore possibilities of inadequate functions without omission.

2. Identify permissions for the code:
   By looking up the deployment, we can know the location of the code. With the location, we can identify the permissions for the code.
3. Enumerate enabled functions by the code:
   By filtering out the functions that can not be executed from the functions enumerated in the step 1, we can enumerate enabled functions.

This algorithm is almost the same as checking algorithm in [2]. The checking algorithm only reports whether all functions can be enabled functions or not, but this derivation algorithm reports the list of enabled functions.

Table 1 is the enabled functions table derived from an environment in Fig. 3 with a policy description in Fig. 4. Because the number of targets and the number of actions in a policy description are finite and their kind is fixed, the size of functions tables are finite and the variation of values in each cell is also finite in general.

4. Which Functions are Required by Users?

In this section, we discuss the role of goals and requirements, and show how to derive required functions from a goal hierarchy.

4.1 Software Requirements Specification for Mobile Code Application

We think mobile code applications are useful in the domains of e-commerce, e-learning, network-game and so on. Their common characteristics are as follows. First their services are dynamically changed therefore their requirements are also dynamically changed. Second, they need scalability. Third, security issues are important for them.

As mentioned in IEEE standard [6], 'software requirements specification (SRS) is a specification for a particular software product, program, or set of programs that performs certain functions in a specific environment'. In the case of SRS for mobile code application, a specific environment can be defined by functions of mobile codes, their deployment and security policies mentioned in Sect. 3.1. Such environment defines what can be performed under the environment, and we represent such things as enabled functions.

4.2 Goals and Required Functions

We assume software requirements are derived from goals of application users. Unfortunately, their goals are not fully achieved by the application under a specific environment in general. So, we should clarify differences between requirements and their goals. In this paper, we regard goals as to-be or ideal goals [7], but we regard requirements as required functions that are required by goals and that can be performed under a given environment.

Because we should test each required function whether it can be performed or not under a given environment, required functions should be represented near in implementation level. Therefore, we should fully decompose or refine the goals so that we can derive required functions, that can be tested in a given environment. The notation of a goal hierarchy is suitable for this purpose because we can convert abstract goals into concrete goals stepwise.

Because security issues are important for mobile code application, we should explicitly handle them. Non-Functional Requirements (NFR) types [8] can be used to identify such issues in top down way. In our method, we mainly develop goal hierarchy in bottom up way by identifying the differences between enabled functions and required functions. Inference rules for goal decomposition [9] will also help us to make goal hierarchy in both ways.

5. Trade-off Analysis

In this section, we show how to identify and compromise the differences between enabled and required functions.

5.1 Identifying the Differences

Using the tabular form for functions, we find conflicts between enabled and required functions tables. Because each row in tables shows each valuable resources and each column shows each code, we try to find conflicts in each cell respectively. We call a set of functions in a cell of a required function table as $RSet$, and a corresponding set in an enabled function table as $ESet$.

1. For each `function+` $\in ESet$
   a. If `function-` $\in RSet$, there is a conflict about `function`.
   b. Else if `function?` $\in RSet$, there can be conflict or vagueness. We should check it manually. We sometimes find unstated conflicts about `function`.
   c. Else there is neither conflict nor vagueness about `function`.
2. For each `function-` $\in ESet$
   a. If `function+` $\in RSet$, there is conflict and goals related to `function` are not satisfied now.
   b. Else if `function?` $\in RSet$, there can be conflict or vagueness. We should check it manually.
   c. Else there is neither conflict nor vagueness about `function`.

We sometimes call each cell by using row and line labels. For example in Table 1, Write-home cell is filled in '`r+ w+`'.

Institute of Electronics, Information, and Communication Engineers

5.2 Compromising the Differences

Here we show the procedure to compromise the differences between an environment and a goal hierarchy, and to get bases of software requirements specification.

1. Write the environment description and the goal hierarchy. We may consult the system requirements specification [10] when we write them.
2. Derive required functions and enabled functions:
   In the first step, we construct required functions from the goal hierarchy. We also construct enabled functions from the environment systematically as mentioned in Sect. 3.2. We should refine the goal hierarchy so as to identify required functions and so that the terms in goals can fit for terms in enabled functions.
3. Identify the differences between required functions and enabled functions:
   Because they are written in the same form and terms in required functions are fit for terms in enabled functions by goal decomposition and refinement, we can systematically identify their differences.
4. Resolve conflicts between enabled functions and required functions: There are two ways to resolve such conflicts.
   - Modify environment:
     So as to meet required functions and goals, the environment is modified if possible. It is relatively easy to modify policies because policies are located in the user's machine. Such modification sometimes enables other functions performed by other mobile codes, so we should check such kind of side effects. It is not easy to modify the deployment and the codes themselves because they are defined by code providers and they are sometimes shared several applications and/or projects.
   - Modify required functions:
     The environment sometimes can not be modified as mentioned just above. In such a case, we should abandon some part of required functions so as to resolve conflicts. As a result, several goals can not be satisfied. In our method, there is no way to recover such things. Our method only enables us to record gaps between goals and required functions, so as to recover them when the environment will be changed in the future.
5. Clarify vagueness of required functions and goals:
   By observing the differences between enabled and required functions, we can sometimes detect requirements that are unstated but should be specified. Such detection enables us to find implicit goals, and to add such goals into the goal hierarchy.
6. Iterate above steps so as to make required functions be consistent with enabled functions. When the environment description and/or the goal hierarchy are changed, inconsistency can be occurred.
   Rename required or enabled functions as compromised functions and finish this procedure, when required functions become consistent with enabled functions.

Currently, we use simple GORA, because we do not handle conflicts among stakeholders. We will use extended version of GORA e.g. [11], when we handle such conflicts.

6. Examples

In this section, we will show examples of an application for e-learning, to demonstrate the usefulness of our method. Outline of goals for the application are as follows.

> We want to provide e-learning system for people over the internet. When learners use our system, each learner solves questions respectively, and the answers are scored respectively. Learners can give remarks to any other learners so as to encourage the progress and the motivation of their study. Because of the scalability of the services, the services should be provided by decentralized system. The service should be also constructed by mobile codes that are already developed by our related company.

We decide to develop this application not as server-side system like CGI but as client-side system using the following mobile codes according to the goals.

We can use the following mobile codes to develop this application.

- Staff.class including a function to mark answers.
- CoLearner.class including a function to give remarks.
- Learner.class including functions to write answers and to see their scores and remarks.

Because each code has not only function(s) above but also other functions and we can not modify codes themselves, we should restrict some of their functions in the client-side using security policies.

We identify the following valuable resources (files) stored in each client machine used by each learner.

- Answers for a question.
- Score for each answer.
- A set of remarks for the question.

For simplicity, we handle only one question, one answer and one set of remarks for the question in this example. The environment for this application is shown in Fig. 5.

6.1 Modifying Policies

In this first example, we will show how to modify policies so as to meet the goals. In addition, we will show how to clarify the goals during the modification.

Institute of Electronics, Information, and Communication Engineers

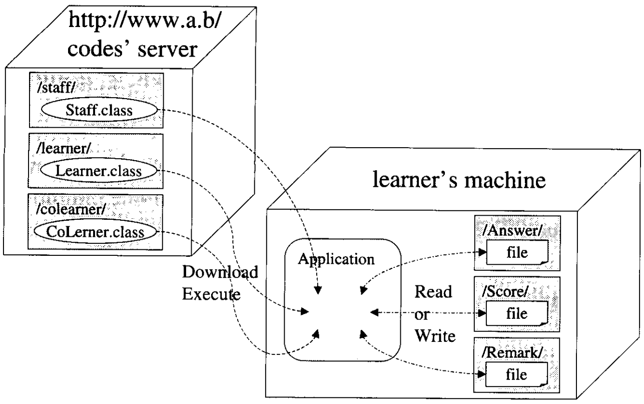


Fig. 5 Environment for this example application.

```
grand codeBase "http://www.a.b/" {
  permission java.io.FilePermission
             "/*", "read, write";
}
```

Fig. 6 Initial policy.

Table 2 Enabled functions table under the initial policy.

	Staff	CoLearner	Learner
Answer	r+ w+		
Score			
Remark			

6.1.1 Initial Policies for the Application

To enable the functions performed by the mobile codes above, we first give the policy in Fig. 6 to the application. Because the policy in Fig. 6 is large enough to enable the codes in Fig. 5 to read and write any files in the learner's machine, we may regard this policy as the initial one. We derive the enabled functions table in Table 2 from the environment in Fig. 5 by using the algorithm in Sect. 3.2. the enabled functions seems to satisfy goals for the application mentioned above. Note that '`r+ w+`' in this Table is of course an abbreviation for 'read, write'.

6.1.2 Goals for the Application

To clarify the required functions of this application, we write a goal hierarchy as shown in Fig. 7. In this figure, thick ovals show goals that are directly related to required functions. As a result, we find required functions table as shown in Table 3.

6.1.3 Conflicts between the Policies and the Requirements

We try to find the conflicts between the policies and the requirements by comparing Tables 2 and 3 using the steps in Sect. 5.1. We find two conflicts about Staff-Answer cell and Learner-Score cell using step 1.b in Sect. 5.1.

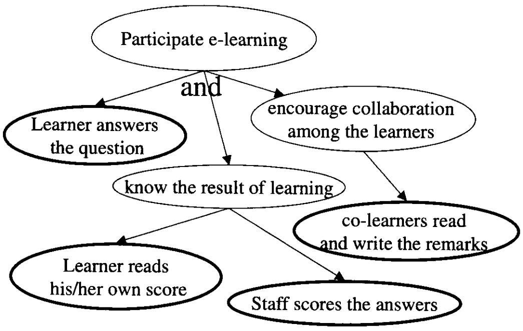

Participate e-learning
and
Learner answers the question
encourage collaboration among the learners
know the result of learning
co-learners read and write the remarks
Learner reads his/her own score
Staff scores the answers

Fig. 7 Initial goal hierarchy.

Table 3 Initial required functions table.

	Staff	CoLearner	Learner
Answer	● r+ w?	?	r+ w+
Score	r+ w+	?	● r+ w?
Remark	?	r+ w+	?

Conflicts with enabled functions

We resolve the conflicts so that required functions are satisfied and that enabled functions are modified. As a result, a staff do not write the answer and a learner do not write score respectively. According to the resolution, we modify the policies as shown in Fig. 8. Because we don't have handle vagueness in this stage yet, policies for CoLearner.class are not changed in Fig. 8.

6.1.4 Vagueness in the Requirements

Several cells in Table 3 are '?'. We regard cells with ? as the symptom of the vagueness of the requirements. We try to investigate and decide the values for such cells, so as to clarify the requirements and their goals.

- As we decide that teachers may give remarks as well as colearners, the value of Staff-Remark cell is to be '`r+ w+`'.
- Because we decide that the answer and its score should not be read by colerners, so as to prevent iniquities such as cheating, and so as to protect the privacy of the learner, the values of CoLearner-Answer cell and CoLearner-Score cell are to be both '`-`'.
- As we decide that the learner himself may give remarks as well as colearners, The value of Learner-Remark cell is to be '`r+ w+`'.

6.1.5 Update the Requirements and Their Goals

As the result of analysis up to here, we have compromised functions table as shown in Table 4.

We can also clarify the goals of the application as

```
grand codeBase
      "http://www.a.b/staff/" {
  permission java.io.FilePermission
             "/Answer/*", "read";
  permission java.io.FilePermission
             "/Score/*", "read, write";
  permission java.io.FilePermission
             "/Remark/*", "read, write";
}
grand codeBase
      "http://www.a.b/learner/" {
  permission java.io.FilePermission
             "/Answer/*", "read, write";
  permission java.io.FilePermission
             "/Score/*", "read";
  permission java.io.FilePermission
             "/Remark/*", "read, write";
}
grand codeBase
      "http://www.a.b/colearner/" {
  permission java.io.FilePermission
             "/*", "read, write";
}
```

Fig. 8 Policies for resolving the conflicts.

Table 4 Compromised functions table.

	Staff	CoLearner	Learner
Answer	r+ w-	-	r+ w+
Score	r+ w+	-	r + w-
Remark	r+ w+	r+ w+	r+ w+

shown in Fig. 9. Dashed ovals in this figure show already existing goals, and other ovals show new added goals. In this figure, we add links between the goal hierarchy and requirements, so as to identify the reason why new goals are added.

6.2 Abandoning Requirements

In this second example, we will show how to abandon some parts of requirements so as to meet policies. Because we maintain the traceability between to-be goals and modified requirements, we can easily identify why and how to compromise the requirements.

6.2.1 Additional Requirements and Their Goals

We have the following additional requirements to extend our business.

> We try to extend our e-learning system so as to certify a credit of a course. In the future, such credit will become compatible with the credit issued by universities.

Table 5 Required functions table derived from Fig. 10.

	Teacher	Admin.	CoLearner	Learner
Answer	r+ w-	-	-	r+ w+
Score	r+ w+	r+ w-	-	r+ w-
Remark	r+ w+	?	r+ w+	r+ w+

Table 6 Enabled functions table constrained by code packaging.

	both in Staff.class			
	Teacher	Admin.	CoLearner	Learner
Answer	r+ w-	r+ w-	-	r+ w+
Score	r+ w+	r+ w+	-	r+ w-
Remark	r+ w+	r+ w+	r+ w+	r+ w+

Based on this statement, we build the goal hierarchy shown in Fig. 10, We replace a goal 'others can not read his score' in Fig. 9 with the goal 'co-learners can not read his score', because an administrator who issues the credit is also a part of others, but the administrator should be able to read the score of course.

We also clarify the required functions table shown in Table 5. We rename 'staff' to 'teacher' so as to distinguish teachers and administrators.

6.2.2 Constraints of Code Packaging

Although we identify new requirements shown in Table 5, we find we can not satisfy the requirements under the given environment. The reason is that both functions for teachers and those for administrators are packaged into the same mobile code 'Staff.class' and we can not take the code into pieces.

As a result, enabled functions table by the codes are as shown in Table 6. We encounter conflicts between requirements and the environment again. We can find the conflicts by comparing Tables 5 and 6. In this case, the features of codes in the environment cause the conflicts.

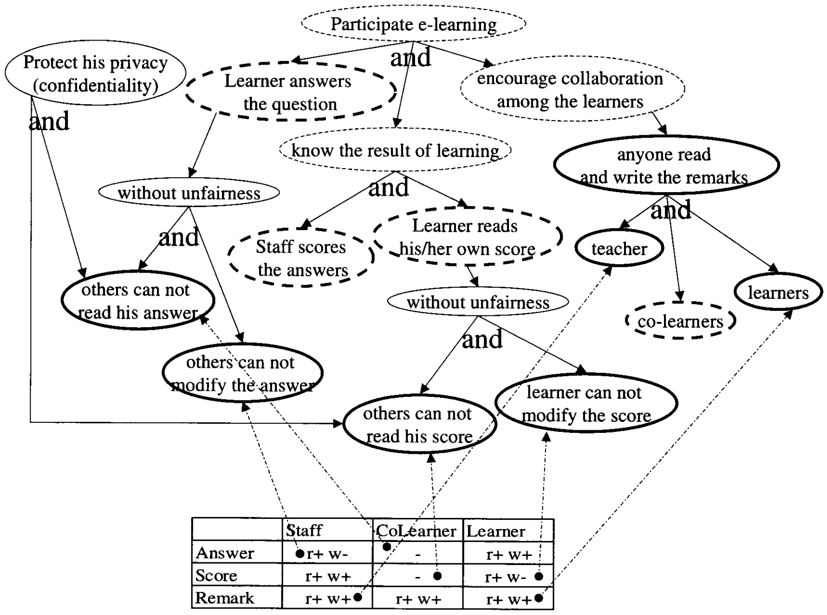
Because we can not change the features, there is no other way to select functions in Table 6 as the compromised functions for our new application.

6.2.3 Traceability about the Compromise

Because of the constraints by the environments, in this case, constraints by the features of a code, we have abandoned several parts of goals, 'protect his privacy including his answer' and 'prevent the unfairness while answering the question'. We have found an abandoned goal 'scoring without unfairness', because our new requirements enable administrators to modify scores unfairly. We add this new goal into Fig. 11.

Because administrators are trustworthy enough to read such information in general, changes of the requirements here are not so serious. However, we should remember and maintain that the changes do not meet original goals of the application. In our method, we do not modify goal hierarchy according to such changes of requirements, but we leave the gaps between the goal hierarchy and the requirements. Such

Institute of Electronics, Information, and Communication Engineers



	Staff	CoLearner	Learner
Answer	●r+ w-	● -	r+ w+
Score	r+ w+	- ●	r+ w- ●
Remark	r+ w+●	r+ w+	r+ w+●

Fig. 9 Updated goal hierarchy.

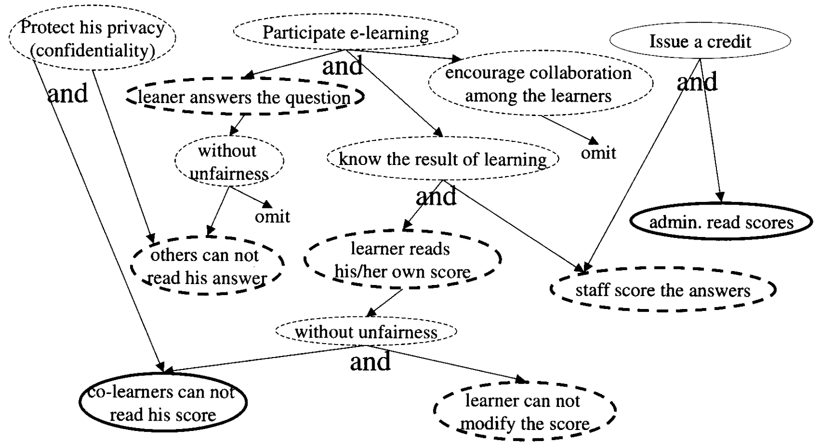
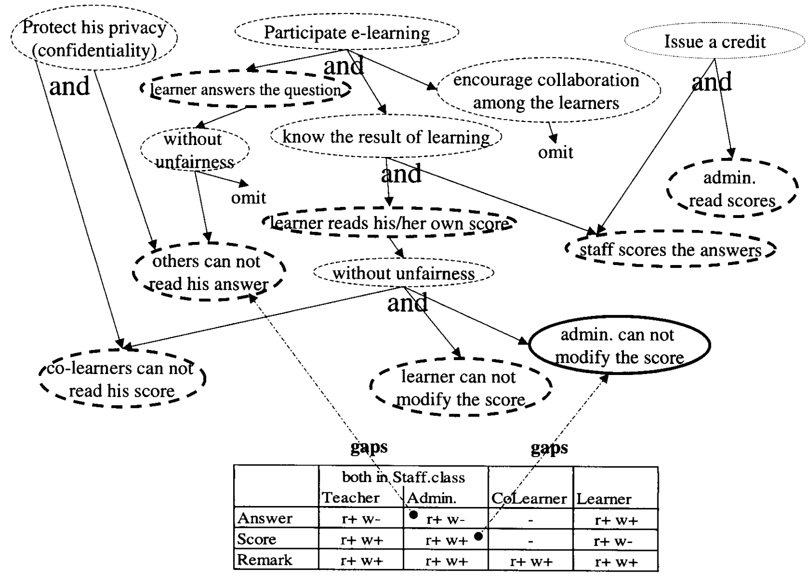


Fig. 10 Goal hierarchy with certification.



	both in Staff.class			
	Teacher	Admin.	CoLearner	Learner
Answer	r+ w-	● r+ w-	-	r+ w+
Score	r+ w+	r+ w+ ●	-	r+ w-
Remark	r+ w+	r+ w+	r+ w+	r+ w+

Fig. 11 Goal hierarchy and compromised functions table.

NII-Electronic Library Service

gaps are written as shown in Fig. 11 so as to remember and maintain the fact.

7. Related Work

Necessity for security requirements engineering is argued in [12]. In the article, six problems are proposed and security policy is mentioned in half of them. The problems related to the security policies are as follows.

- Threats and anti-requirements
- Security policy and why it is often ignored
- Where is the organization?

We discuss advantages with the context of first two problems, an limitation with the context of last problem. We also discuss other advantages and limitations referring other articles.

7.1 Advantages

First, we talk about anti-requirements, that are the requirements of malicious users such as intruders. For finding anti-requirements, misuse cases and negative scenario are used [13], [14]. Even if we use misuse case and negative scenario, it does not seem to be easy to imagine anti-requirements. In our method, we can directly face the functions that can be used for anti-requirements from enabled functions, because enabled functions directly show the possibilities of inadequate and/or malicious functions. So it is slightly easy to imagine anti-requirements. Goal oriented techniques are also applied in this area [15]. We also think taxonomy for NFR such as security requirements is useful for exploring anti-requirements.

Next, we talk about organizational security policies. In [16], Levels of abstraction of security policies are categorized into four levels; organizational requirements, computer policy models, access control models and implementation models. Organizational security policies are studied in [17], [18], where goal oriented analysis is also used and useful taxonomy for privacy policy is given. On the other hand, our study focus on the lower levels because of the features of our application domains. In [12], it is cited that 'most issues for security policies are derived from the solution world rather than the problem world, and do not encourage a systematic exploration of the issue'. Although our approach deeply depends on the problem world, we believe it encourages a systematic exploration. The reason is that we only focus on applications using mobile codes, and we can not distinguish between requirements and implementations because such applications are integrated and executed on the fly along with requirements in the moment. Of course, security issues for mobile codes are widely focused [19], but relationships between them and requirements are rarely discussed.

For resolving conflicts between an environment including policies and requirements, we have options to abandon some part of requirements or to modify policies. However, we have another option, that is, to use other mobile codes, which support required functions and which can work completely and adequately under the environment. We can also choose this option if we have some kind of directory services for mobile codes. Our method can be used to explore suitable codes among the libraries in such directory.

7.2 Limitations

A problem 'where is the organization?' in [12] is about security requirements about 'who runs the application?'. This problem is related to the role based access control and it is important in a multi-user environment. This is also important when a user of an application himself also moves from one machine to another. In our method, we only handle simple single-user application mentioned in Sect. 3.1, and don't handle neither a multi-user environment nor mobility of an user.

Java security system is already extended in this view point [20]. The extended part is called *Java Authentication and Authorization Service* (JAAS) framework, and it enables us to handle a security issue about 'who runs the code'. In the framework, an access control mechanism based on the fact that who runs the code is called *user-centric style*. Another access control mechanism based on the fact that which codes are used is called *code-centric style*. If we use both mechanisms, we can provide a multi-user environment within a JVM. This is similar to ordinary multi-user operating systems like UNIX. Although only the code centric mechanism is focused in our method and this means we only handle the single user environment like MS-DOS, we don't think this limitation becomes serious problem. The reasons are that most Java applications are used for the single user, and that a multi-user system is normally constructed with several Java applications that are connected by communication mechanism e.g. RMI.

As already mentioned previous section, goal oriented techniques are widely used for security requirements [15], [18], [21]. We have also our own goal oriented method AGORA [11]. We use the most simple notation for goal hierarchy in this paper, because we don't handle conflicts among multiple stakeholders. When we handle such conflicts, we can use our AGORA notation instead of current one, because AGORA has a mechanism for representing preferences of multiple stakeholders.

In this paper, we only handle an application using Java mobile codes. However, we download unknown programs and/or libraries from Internet and use it without care in every day. Therefore, we should extend domains of our method to codes in general. Concept for our work can be extended, but a notation describing security policies is too poor to be used in codes in general. we need more expressive language for representing policies, e.g. Ponder [22].

Ponder can express access policies to prohibit some functions, while Java2 security policy can represent policies to allow some functions only. When we introduce former kind of negative policies, we should modify the derivation

Institute of Electronics, Information, and Communication Engineers

algorithm for enabled functions in Sect. 3.2 but we do not need to modify the tabular form for functions. We also meet conflicts among a policy description when we use both negative and positive policies at the same time. We can identify such conflicts syntactically by checking such description, but we cannot resolve the conflicts systematically. We need dependencies among the targets(valuable resources) when we identify such conflicts. Typical example is an inclusion relationship among folders. Current version of our method does not need to consider such dependencies for conflict detection, but policy description could be redundant without such dependencies.

8. Conclusions and Future Works

In this paper, we propose a method for analyzing trade-off between security policies for Java mobile codes and requirements for Java application. By using our method, we can get requirements which can be realized in given environment. We also identify which initial requirements are abandoned so as to meet the environment. This is achieved by constructing a goal hierarchy for ideal application. Our method is partially supported by our tool for checking and generating Java security policies.

Applications using mobile codes seems to be ruled by some specific software architectures, and security issues for such application seems to be deeply related to the architectures. Therefore, relationships between requirements and software architectures will be useful when we think about the relationships between requirements and security policies. We can find many papers about relationships between requirements and software architectures [23], but we can not find issues for security policies among them. We want to investigate such meta-relationships in the future.

Acknowledgments

This work has been supported by Grant-in-Aid for Young Scientists (B) (KAKENHI #5700028), the Ministry of Education, Culture, Sports, Science and Technology, Japan. The authors would like to thank Dr. Shin Nakajima, Hosei University for insightful comments to our work.

References

[1] A. van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," RE'01, pp.249–263, Aug. 2001.

[2] H. Kaiya, H. Furukawa, and K. Kaijiri, "Security policy checker and generator for Java mobile codes," Engineering Information Systems in the Internet Context (EISIC), pp.255–264, IFIP TC8/WG8.1, Kluwer Academic Publishers, Sept. 2002.

[3] T. Thorn, "Programming languages for mobile code," ACM Computing Surveys, vol.29, no.3, pp.213–239, Sept. 1997.

[4] S. Oaks, Java Security, O'Reilly, 2nd ed., 2001.

[5] JODE decomplier homepage. http://jode.sourceforge.net/, Feb. 2003.

[6] IEEE Recommended Practice for Software Requirements Specification, Oct. 1998. IEEE Std 830-1998, ISBN 0-7381-0332-2 SH94654(Print).

[7] E. Kavakli, "Goal-oriented requirements engineering: A unifying framework," Requirements Engineering, vol.6, pp.237–251, 2002.

[8] L. Chung, B.A. Nixon, E. Yu, and J. Mylopoulos, Non-functional Requirements in Software Engineering, Kluwer Academic Publishers, 2000.

[9] A. van Lamsweerde, R. Darimont, and E. Letier, "Managing conflicts in goal-driven requirements engineering," IEEE Trans. Softw. Eng., vol.24, no.11, pp.908–926, Nov. 1998.

[10] IEEE Guide for Developing System Requirements Specifications, Dec. 1998. IEEE Std 1233-1998, ISBN 0-7381-0337-3 SH94654(Print).

[11] H. Kaiya, H. Horai, and M. Saeki, "AGORA: Attributed goal-oriented requirements analysis method," IEEE Joint International Requirements Engineering Conference, RE'02, pp.13–22, Sept. 2002.

[12] R. Crook, D. Ince, L. Lin, and B. Nuseibeh, "Security requirements engineering: When anti-requirements hit the fan," IEEE Joint International Requirements Engineering Conference, RE'02, pp.203–205, Essen, Germany, Sept. 2002.

[13] I. Alexander, "Initial industrial experience of misuse cases in trade-off analysis," IEEE Joint International Requirements Engineering Conference, RE'02, pp.61–68, Essen, Germany, Sept. 2002.

[14] G. Sindre and A.L. Opdahl, "Templates for misuse case description," REFSQ'2001 Proceedings, pp.125–136, 2001.

[15] L. Chung, "Dealing with security requirements during the development of information systems," CAiSE'93 Proceedings, pp.234–251, 1993.

[16] R.K. Thomas and R.S. Sandhu, "Conceptual foundations for a model of task-based authorizations," Proc. IEEE Computer Security Foundation Workshop VII, pp.66–79, 1994.

[17] A. Anton, J. Earp, C. Potts, and T. Alspaugh. "The role of policy and stakeholders privacy values in requirements engineering," RE'01 Proceedings, pp.138–145, 2001.

[18] A. Anton, J. Earp, and A. Reese, "Analyzing website privacy requirements using a privacy goal taxonomy," IEEE Joint International Requirements Engineering Conference, RE'02, pp.23–31, Essen, Germany, Sept. 2002.

[19] A.D. Rubin and D.E. Geer, Jr., "Mobile code security," IEEE Internet Computing, vol.2, no.6, pp.30–34, Nov. and Dec. 1998.

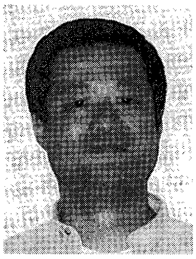
[20] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers, "User authentication and authorization in the Java^{TM} platform," Proc. 15th Annual Computer Security Applications Conference, pp.285–290, Dec. 1999.

[21] A. van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," IEEE Trans. Softw. Eng., vol.26, no.10, pp.978–1005, Oct. 2000.

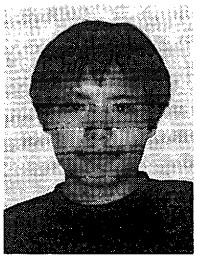
[22] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder specification language," Proc. Policy 2001: Workshop on Policies for Distributed Systems and Networks, pp.18–39, Jan. 2001.

[23] STRAW'01, First International Workshop From Software Requirements to Architectures homepage. http://www.cin.ufpe.br/~straw01/, May 2001.

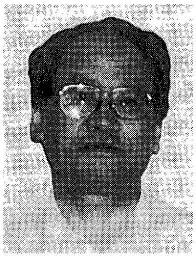
Institute of Electronics, Information, and Communication Engineers



Haruhiko Kaiya is an associate professor of Software Engineering at Shinshu University, Japan.
http://www.cs.shinshu-u.ac.jp/ kaiya/



Kouta Sasaki is a graduate school student of Software Engineering at Shinshu University, Japan.



Kenji Kaijiri is a professor of Software Engineering at Shinshu University, Japan.