Trade-off Analysis between Security Policies for Java Mobile Codes and Requirements for Java Application

Haruhiko Kaiya   Kouta Sasaki   Yasunori Maebashi   Kenji Kaijiri
Dept. of Computer Science, Shinshu University
Wakasato 4-17-1, Nagano 380-8553, Japan
kaiya@cs.shinshu-u.ac.jp   http://www.cs.shinshu-u.ac.jp/˜kaiya/

1. Introduction

We propose a method for analyzing trade-off between security policies for Java mobile codes and requirements for Java application. We assume that mobile codes are downloaded from different sites, they are used in an application on a site, and their functions are restricted by security policies on the site. We clarify which functions can be performed under the policies on the site using our tool[1]. We also clarify which functions are needed so as to meet the requirements for the application by goal oriented requirements analysis(GORA).

By comparing functions derived from the policies and functions from the requirements, we can find conflicts between the policies and the requirements, and also find vagueness of the requirements. By using our tool and GORA again, we can decide which policies should be modified so as to meet the requirements. We can also decide which requirements should be abandoned so as to meet policies which can not be changed.

2. Overview of our Method

In Figure1, we show components that are used in our method and their relationships. Using this figure, we will show the procedure how to analyze the trade-off between environments and requirements.

The inputs of our method are an environment and a goal hierarchy. The environment consists of mobile codes, their deployment over the network and security policies for a site where intended application will be executed. The goal hierarchy shows what application users want because we regard goals as to-be or ideal goals in this paper.

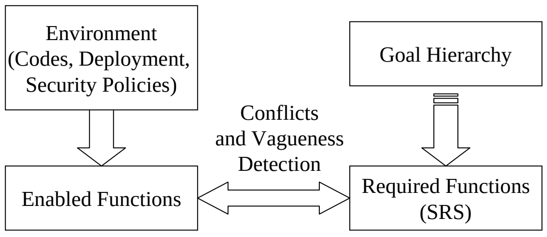


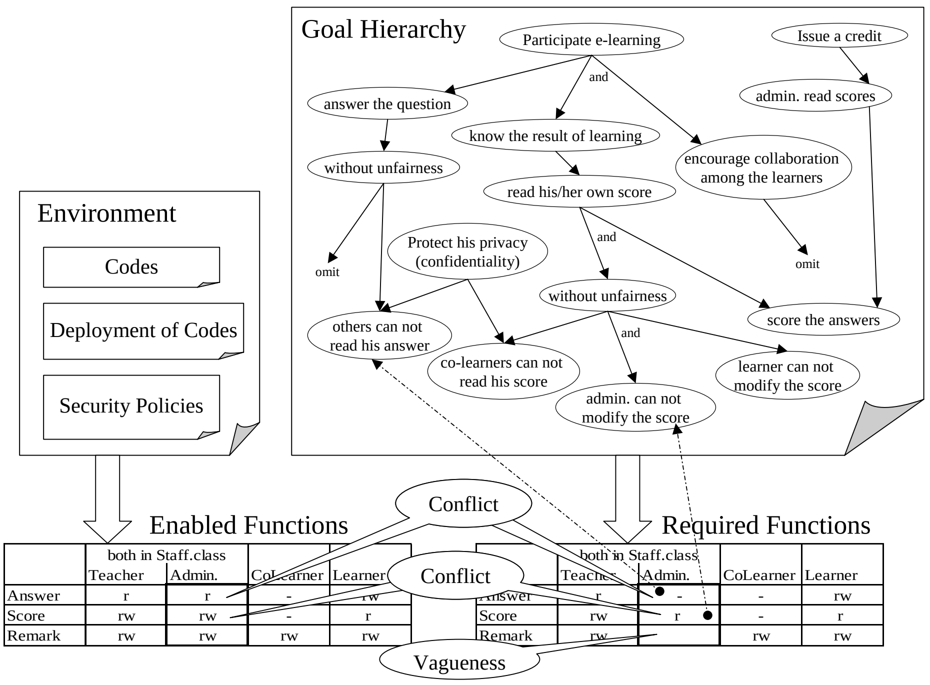
Figure 1. Components in our Method

The main output of our method is required functions, that become main parts of software requirements specification(SRS). Each function represents a permission for someone or some code to do something. At the end of the analysis process, our method guarantees required functions to be feasible under the given environment, but the functions are not always consistent with the goal hierarchy.

The goal hierarchy is sometimes modified and/or extended because implicit and/or unidentified goals are found by using this method. The environment is also sometimes modified so as to meet the goal hierarchy in Figure1. As a result, the goal hierarchy becomes clearer, and the environment becomes fit to the hierarchy if possible.

Here is the procedure of our method.

1. Get the description of the environment and the goal hierarchy.
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 because we can identify which functions can be performed or not. Note that required and enabled functions are represented in tabular form.
3. Identify the differences between required functions and enabled functions:
   Because they are written in the same form, 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. It is easy to modify policies because policies are located in the



Enabled Functions

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

Required Functions

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

Figure 2. Snapshot of an Analysis for an E-learning System

user's machine. Such modification sometimes enables other functions performed by other mobile codes, so we should check such kind of side effects by using our tool. It is not easy to modify the deployment because it is defined by code providers and such deployment is sometimes shared several applications and/or projects. We assume that we can not modify the codes directly in this paper.

Modify required functions: The environment sometimes can not be modified as mentioned just above. In such a case, we should abandon some part of requirements so as to resolve conflicts. As a result, several functions expected by the application users are not provided. 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 complete required functions if the environment and/or the hierarchy are changed.

Currently, we use simple GORA, because we do not handle conflicts among stakeholders. We will use extended version of GORA e.g.[2], when we handle such conflicts.

3. Example

We will show a part of an example to demonstrate the usefulness of our method. In this example, requirements for an e-learning system are analyzed. A company required this system will issue the credit of courses by the system. Such credit can be compatible with the credit issued by an university.

Figure2 shows a snapshot of an analysis for this system. As the result of GORA, several goals including two goals, 'others can not read one's answer' and 'administrator can not modify the score', are found. Based on the goals, we have derived required functions. We also derive enabled functions from the environment, and we know we can not modify the environment. By comparing two kinds of functions, we find two conflicts and one vagueness.

In this case, we should abandon these two goals because conflicts are related to the goals and we can not modify enabled functions under this environment. As a result, current enabled functions are selected as new required functions, then we record gaps between new required functions and original goals. This record will contribute future evolution of this system and the environment.

References

[1] H. Kaiya, Furukawa, and K. Kaijiri. Security Policy Checker and Generator for Java Mobile Codes. IFIP TC8/WG8.1 Working Conference on EISIC, pages 255–264, Sep. 2002.

[2] H. Kaiya, H. Horai, and M. Saeki. AGORA: Attributed Goal-Oriented Requirements Analysis Method. Proc. of RE'02, pages 13–22, Sep. 2002.