Regular Paper

A Flexible Execution Control Method of Application Software for Educational Windows PCs

Keita Kawano[1,a)] Daisuke Okamoto[1] Masanori Fujiwara[1,†1] Nariyoshi Yamai[1]

Received: June 28, 2013, Accepted: December 4, 2013

Abstract: Recently, many educational institutions have had to maintain a large number of PCs (educational PCs). It is common for the administrators of these institutions to use a disk image distribution system to manage them all together. It allows the administrators to keep all of their educational PCs in the same configuration with ease. This method, however, robs them of the flexibility of management. Suppose that use of certain application software is restricted to specific users or sites in those institutions. Suppose also that use of other application software is requested to be prohibited by teachers during classes. It is hard for the administrators to satisfy all of these requirements with the traditional method since it causes heavy administrative burden. This paper proposes a flexible method to manage application software on educational Windows PCs without significant efforts of the administrators even with these requirements. The proposed method controls the execution of individual application software on each educational PC to create the requested environments. Teachers as well as the administrators can directly and dynamically change its configuration during their classes. An execution control system was actually implemented as a prototype to show the feasibility of the proposed method.

Keywords: educational PC, application software, execution control, management cost, flexibility

1. Introduction

Recently, to meet the requirements for information literacy education and the demands for computer-assisted teaching, many educational institutions such as universities have had to support a lot of educational PCs [1], [2], [3]. Since an increase in the number of educational PCs raises the administrative burden, a method to alleviate it has been a major concern of the administrators in these institutions [4], [5].

Disk image distribution systems are usually used for this purpose. With a disk image distribution system, a disk image of one model PC can be distributed to all of the educational PCs at once. This helps the administrators keep all of their educational PCs in the same configuration.

This method, however, robs them of the flexibility of management that is always required in most higher educational institutions like universities. There, each department has its own educational policies. This means that each department usually requests different environments for their educational PCs. Most of all, application software environments for its education are likely to differ among departments.

Some departments may request certain application software whose license fee needs to be paid. Other departments may hope to use other application software that should be used after sufficient ethics and compliance education and thus the administrators do not want the other students to use it. Even in classes, teachers hope to restrict use of specific application software during examinations.

With the traditional method using a disk image distribution system, it is hard for the administrators to satisfy all of these requirements. The first requirement, which causes fruitless license fee, can be satisfied if *user license* (the amount of payment depends on the number of users) options are provided for the application software. However, if no sophisticated license servers are provided, students who do not belong to the corresponding departments can waste the limited number of licenses. In addition, there is application software, especially from small software vendors or individuals, for which only *device license* (the amount of payment depends on the number of installations) options are provided.

The first and second requirements might be satisfied to some extent if the administrators could maintain a different model disk image for each request. The number of installations of the application software could be reduced to the number of educational PCs in the PC rooms of the corresponding departments. The application software that should be used after sufficient ethics and compliance education could be installed only within the PC rooms of the corresponding departments.

This method, however, is not acceptable if many departments have different demands since the administrative burden increases linearly with the number of managed model disk images. Moreover, if use of certain application software is asked to be restricted only to a group of users such as a group of students taking a specific class, the method using multiple model disk images cannot fulfill this request depending on the way of image distribution, as described later in Section 4.1. Even though it is possible, it raises the number of managed model disk images, increasing the administrative burden. The method also does not have real-time control

[1] Okayama University, Okayama 700–8530, Japan
[†1] Presently with JR WEST IT Solutions Company
[a)] keita@cc.okayama-u.ac.jp

of restriction policies to support the third requirement. Hence, a method to regain the lost flexibility of management due to the use of disk image distribution systems without committed efforts of the administrators even with the above requirements is needed.

This paper proposes a flexible method to manage application software on educational Windows PCs [*1]. The proposed method creates individual application software environments by controlling the execution of application software on each educational PC. It lets the administrators restrict use of each application software to specific groups of users or educational PCs without relying on creating specific model disk images, which increases the administrative burden linearly as described above. Teachers as well as the administrators are allowed to build arbitrary application software environments directly and dynamically during their classes without losing definitive decisions of the administrators.

Note that there are several options for operating systems on educational PCs, such as the Microsoft Windows family, Apple OS X, and Linux [9], [10], [11], [12]. In addition, some institutions provide different versions of the same operating system due to the constraints of indispensable application software while other institutions offer different families of operating systems. This paper, however, assumes that only a single version of Windows is installed on all of the educational PCs for implementation reasons and for simplicity [*2]. This paper also assumes that no multiboot feature is used on the educational PCs, also for simplicity.

The rest of this paper is organized as follows. The management cost of educational PCs is formulated in Section 2. Section 3 discusses related work. Section 4 describes an overview of the management of educational PCs using a disk image distribution system and its problem is clarified. A flexible management method with the execution control of application software is proposed in Section 5. An implementation of the proposed method is described in Section 6. Section 7 concludes this paper.

2. Management Cost of Educational PCs

In this section, the management cost of educational PCs in a typical educational institution is formulated to prepare for the discussions in this paper. It is assumed that only a single version of Windows is installed on all of the educational PCs and the multiboot feature is not used, as described in the previous section.

The number of educational PCs n_{pc}, the number of their users n_{user}, and the number of PC rooms in that institution are assumed to be fixed and given, for simplicity. Variables depending on these fixed variables, such as license fees depending on n_{pc}, are thus fixed too, if they do not depend on any of the other variables. In addition, demands for use of each application software (who uses it or where it is used) are also assumed to be fixed and given though the administrators cannot always meet them.

Basically, the management cost c of educational PCs is expressed as the sum of the three types of costs as follows:

$$c = c_{\mathrm{s}} + c_{\mathrm{l}} + c_{\mathrm{o}}, \tag{1}$$

where c_{s} is *system cost*, c_{l} is *license cost*, and c_{o} is *operating cost*.

The system cost is the cost of introducing and maintaining the educational system. It is assumed that software and hardware maintenance contracts are required for the educational system. When using a disk image distribution system, its initial and maintenance costs, including license fee of the system, are contained in this category. The license cost is needed whenever any application software whose license fee needs to be paid is used on the educational PCs.

Moreover, the administrators have a responsibility to keep their educational PCs up-to-date. Whenever any new patches are released, they have to apply them as soon as possible. Whenever teachers ask them to install new application software, they have to do it before their classes. With a disk image distribution system, the administrators have to make these changes to all of the related model disk images and redistribute them. These operations create the operating cost. In our university, we perform these operations once a month, after periodic releases of patches of the operating system. Other operations of the administrators such as system upgrade, setting change, user education and support, and troubleshooting also add to the operating cost. System upgrade cost is generated when a system requires any upgrades, such as security upgrades. Setting change cost is created when the administrators have to change settings of their systems. User education and support, and troubleshooting costs get higher when a system becomes more complex, usually due to integrating several systems.

Since budget and human resources of the ICT department in most universities have continued to decrease in recent years, it is essential to establish a way to provide requested services within the limited resources. To address this issue, the traditional method using a disk image distribution system has been introduced in many universities, reducing c_{o}. We first formulate the management cost of the traditional method with only one model disk image to make clear its characteristics.

The management cost of the traditional method with one model disk image $c_{\mathrm{tra}}(1)$ is expressed as follows:

$$\begin{aligned} c_{\mathrm{tra}}(1) = {} & c_{\mathrm{s,tra}}(1) + \left(\sum_{a \in A_{\mathrm{user}}} n_{\mathrm{user}}(a) c_{\mathrm{l,user}}(a, n_{\mathrm{user}}(a)) \right. \\ & \left. + \sum_{a \in A_{\mathrm{pc}}} n_{\mathrm{pc}} c_{\mathrm{l,pc}}(a, n_{\mathrm{pc}}) \right) \\ & + \left(c_{\mathrm{o,os}} + \sum_{a \in A} c_{\mathrm{o,app}}(a) + c_{\mathrm{o,tra}}(1) \right), \end{aligned} \tag{2}$$

where $c_{\mathrm{s,tra}}(1)$ is the system cost of the traditional method with one model disk image. It contains costs for educational PCs, various servers, storages, and so on. A_{user} and A_{pc} are sets of installed application software with user license options, such as floating license and campus license, and with device license options, such as free license and volume license, in this institution, respectively. A is a set of installed application software in this institution. That is, $A_{\mathrm{user}} \cup A_{\mathrm{pc}} = A$ and $A_{\mathrm{user}} \cap A_{\mathrm{pc}} = \emptyset$. $n_{\mathrm{user}}(a)$ is the number of users for the application software a. That is, $n_{\mathrm{user}}(a) \leq n_{\mathrm{user}}$. $c_{\mathrm{l,user}}(a, n_{\mathrm{user}}(a))$ is the per-user license cost for a with user license options when $n_{\mathrm{user}}(a)$ users use a. $c_{\mathrm{l,pc}}(a, n_{\mathrm{pc}})$ is the per-device license cost for a with device license options when a is installed on

[*1] This work is an extension of some of our previous work [6], [7], [8].

[*2] Actually, the operating system on all of our current educational PCs is Windows 7.

n_{pc} (whole of) educational PCs. $c_{\mathrm{o,os}}$ is the operating cost for updating and testing the operating system on the model disk image. $c_{\mathrm{o,app}}(a)$ is the operating cost for installing, updating, testing, and uninstalling a. $c_{\mathrm{o,tra}}(1)$ is the remaining operating costs of the traditional system with one model disk image. The costs include system upgrades, setting changes, user education and support, troubleshooting, and so on.

Equation (2) shows that the license cost for the application software with device license options increases in proportion to n_{pc} with this method. Since there have been a large number of educational PCs in most universities as described above, this method wastes unnecessary license fees if just a limited group of users, such as students from a department, uses certain application software. As a result, the group usually gives up the use of the application software if the group is asked to pay all of the license fee. Otherwise, the administrators are requested to manage two model disk images: one having the application software and the other not having the application software. Multiple model disk images are also requested, when certain application software that the administrators hesitate to serve for all of their users is required. We thus formulate the management cost of the traditional method with multiple model disk images.

The management cost of the traditional method with n_{m} (≥ 2) model disk images $c_{\mathrm{tra}}(n_{\mathrm{m}})$ is expressed as follows:

$$\begin{aligned} c_{\mathrm{tra}}(n_{\mathrm{m}}) = {} & (c_{\mathrm{s,tra}}(1) + c_{\mathrm{s,tra}}(n_{\mathrm{m}})) \\ & + \left(\sum_{a \in A_{\mathrm{user}}} n_{\mathrm{user}}(a) c_{\mathrm{l,user}}(a, n_{\mathrm{user}}(a)) \right. \\ & \left. + \sum_{a \in A_{\mathrm{pc}}} n_{\mathrm{pc,ins}}(a) c_{\mathrm{l,pc}}(a, n_{\mathrm{pc,ins}}(a)) \right) \\ & + \left(\sum_{m \in M} \left(c_{\mathrm{o,os}} + \sum_{a \in A_{\mathrm{ins}}(m)} c_{\mathrm{o,app}}(a) \right) \right. \\ & \left. + c_{\mathrm{o,tra}}(1) + c_{\mathrm{o,tra}}(n_{\mathrm{m}}) \right), \end{aligned} \quad (3)$$

where $c_{\mathrm{s,tra}}(n_{\mathrm{m}})$ is the additional system cost of the traditional method with n_{m} model disk images. It contains additional processing and storage capacity of servers for disk image distribution or additional servers themselves for disk image distribution, if required. $n_{\mathrm{pc,ins}}(a)$ is the number of educational PCs where the application software a is installed. That is, $n_{\mathrm{pc,ins}}(a) \leq n_{\mathrm{pc}}$. $c_{\mathrm{l,pc}}(a, n_{\mathrm{pc,ins}}(a))$ is the per-device license cost for a with device license options when a is installed on $n_{\mathrm{pc,ins}}(a)$ educational PCs. M is a set of managed model disk images. That is, the number of elements in M is n_{m}. $A_{\mathrm{ins}}(m)$ is the set of installed application software on the model disk image m. That is, $\bigcup_{m \in M} A_{\mathrm{ins}}(m) = A$. $c_{\mathrm{o,tra}}(n_{\mathrm{m}})$ is the additional operating cost of the traditional method with n_{m} model disk images, such as additional system upgrades (if required), setting change, and troubleshooting costs. Other variable definitions are identical with those in Eq. (2). Since this paper assumes that the operating system on all of the educational PCs is a single version of Windows as described above, $c_{\mathrm{o,os}}$ is considered to be fixed, for simplicity, though some efficiency gains may be obtained in fact. Similarly, each $c_{\mathrm{o,app}}(a)$ is considered to be fixed.

Equation (3) shows that the operating costs for maintaining operating systems and application software increase in proportion to n_{m} with this method. Not only budget but also human resources of the ICT department have continued to decrease in recent years as described above. This is not acceptable if many groups of users have different demands. In that case, the administrators have to decline some of those requests for specific model disk images to keep n_{m} small since $c_{\mathrm{o,os}}$ and each $c_{\mathrm{o,app}}(a)$ for each m are concentrated in a specific period of days along with taking a long time for each and thus cause heavy burden on the administrators.

In practice, most application software requested from specific groups of users, especially small groups of users, will not be served as a general rule with both methods due to cost-effectiveness. Our university has this general rule currently. Both methods also do not support real-time control of restriction policies by teachers during their classes.

The method proposed in this paper prevents c_{o} from excessively increasing while providing flexible application software environments including real-time execution control which is not fulfilled with the traditional method. The execution control of application software on each educational PC described later in Section 5 helps this. If due to only the policies in the institution, the administrators do not have to manage multiple model disk images unlike the traditional method. This prevents c_{o} from excessively increasing.

In addition, the proposed method reduces c_{l} if some software vendors with only device license options in their standard (and thus have no special scheme to control the execution of their application software) accept the assumption of the proposed method: just installing certain application software on educational PCs do not consume licenses if its execution is restricted. Multiple software vendors, especially small software vendors or individuals, will be willing to provide the institution with a non-standard contract following the above assumption, as long as reasonable and convincing execution control is performed and they can trust the institution, without passing up on their sales opportunities. In this case, the license cost for certain application software is reduced to the amount of the number of possible execution. Note that the proposed method supports both user and device license options with its execution control mechanism described later in Section 5.

The management costs of the proposed method with one model disk image $c_{\mathrm{pro}}(1)$ and with n_{m} (≥ 2) model disk images $c_{\mathrm{pro}}(n_{\mathrm{m}})$ are expressed as follows:

$$\begin{aligned} c_{\mathrm{pro}}(1) = {} & \left(c_{\mathrm{s,tra}}(1) + c_{\mathrm{s,pro}}(1) \right) \\ & + \left(\sum_{a \in A_{\mathrm{user}}} n_{\mathrm{user}}(a) c_{\mathrm{l,user}}(a, n_{\mathrm{user}}(a)) \right. \\ & + \sum_{a \in A_{\mathrm{pc,ins}}} n_{\mathrm{pc}} c_{\mathrm{l,pc}}(a, n_{\mathrm{pc}}) \\ & \left. + \sum_{a \in A_{\mathrm{pc,exe}}} n_{\mathrm{exe}}(a) c_{\mathrm{l,exe}}(a, n_{\mathrm{exe}}(a)) \right) \\ & + \left(c_{\mathrm{o,os}} + \sum_{a \in A} c_{\mathrm{o,app}}(a) + c_{\mathrm{o,tra}}(1) + c_{\mathrm{o,pro}}(1) \right), \end{aligned}$$

$$\begin{aligned} c_{\mathrm{pro}}(n_{\mathrm{m}}) = {} & \left(c_{\mathrm{s,tra}}(1) + c_{\mathrm{s,tra}}(n_{\mathrm{m}}) + c_{\mathrm{s,pro}}(n_{\mathrm{m}})\right) \\ & + \left(\sum_{a \in A_{\mathrm{user}}} n_{\mathrm{user}}(a) c_{\mathrm{l,user}}(a, n_{\mathrm{user}}(a)) \right. \\ & + \sum_{a \in A_{\mathrm{pc,ins}}} n_{\mathrm{pc,ins}}(a) c_{\mathrm{l,pc}}(a, n_{\mathrm{pc,ins}}(a)) \\ & \left. + \sum_{a \in A_{\mathrm{pc,exe}}} n_{\mathrm{exe}}(a) c_{\mathrm{l,exe}}(a, n_{\mathrm{exe}}(a)) \right) \\ & + \left(\sum_{m \in M} \left(c_{\mathrm{o,os}} + \sum_{a \in A_{\mathrm{ins}}(m)} c_{\mathrm{o,app}}(a) \right) \right. \\ & \left. + c_{\mathrm{o,tra}}(1) + c_{\mathrm{o,tra}}(n_{\mathrm{m}}) + c_{\mathrm{o,pro}}(n_{\mathrm{m}}) \right), \qquad (4) \end{aligned}$$

where $c_{\mathrm{s,pro}}(1)$ and $c_{\mathrm{s,pro}}(n_{\mathrm{m}})$ are the additional system costs of the proposed method with one and n_{m} model disk images, respectively. They contain additional server cost and additional system maintenance cost to improve the proposed system during its operation. $A_{\mathrm{pc,ins}}$ and $A_{\mathrm{pc,exe}}$ are sets of application software, in this institution, with only device license options in its standard. Certain application software is classified into $A_{\mathrm{pc,exe}}$ if its software vendor provides the institution with a non-standard contract following the assumption of the proposed method. Otherwise, the application software is classified into $A_{\mathrm{pc,ins}}$. That is, $A_{\mathrm{pc,ins}} \cup A_{\mathrm{pc,exe}} = A_{\mathrm{pc}}$ and $A_{\mathrm{pc,ins}} \cap A_{\mathrm{pc,exe}} = \emptyset$. $n_{\mathrm{exe}}(a)$ is the number of users who can execute the application software a or the number of educational PCs where a can be executable. That is, $n_{\mathrm{exe}}(a) \leq n_{\mathrm{pc,ins}}(a)$. $c_{\mathrm{l,exe}}(a, n_{\mathrm{exe}}(a))$ is the per-user or per-device license cost for a when $n_{\mathrm{exe}}(a)$ users can execute a or a can be executable on $n_{\mathrm{exe}}(a)$ educational PCs. $c_{\mathrm{o,pro}}(1)$ and $c_{\mathrm{o,pro}}(n_{\mathrm{m}})$ are the additional operating costs of the proposed method with one and n_{m} model disk images, respectively. They contain costs to negotiate with application software vendors and additional system upgrade, setting change, user education and support, and troubleshooting costs. Other variable definitions are identical with those in Eqs. (2) and (3).

Equation (4) shows that the operating costs for maintaining operating systems and application software also increase in proportion to n_{m} with the proposed method with n_{m} model disk images. In addition, if no requested software vendors accept the proposed system, the proposed method cannot save c_{l}. The proposed method, however, reduces n_{m} itself compared to the traditional method when the administrators have to restrict use of each application software to specific groups of users or educational PCs, preventing c_{o} from excessively increasing, as described above. It can save a limited number of licenses for students who actually need the application software. The proposed method can also provide real-time execution control for application software by teachers as well as the administrators without losing definitive decisions of the administrators, as described later in Section 5.3.

In terms of usefulness, $c_{\mathrm{s,pro}}(1)$, $c_{\mathrm{s,pro}}(n_{\mathrm{m}})$, $c_{\mathrm{o,pro}}(1)$ and $c_{\mathrm{o,pro}}(n_{\mathrm{m}})$ have to be kept low against the benefit of the proposed method. We tried to prevent $c_{\mathrm{s,pro}}(1)$ and $c_{\mathrm{s,pro}}(n_{\mathrm{m}})$ from increasing by implementing the system without additional license fees as far as possible, as described later in the next section. The proposed method does not require an additional powerful server according to the results of its fundamental performance evaluations with its prototypes described later in Section 6.4. We also tried to prevent setting change cost in $c_{\mathrm{o,pro}}(1)$ and $c_{\mathrm{o,pro}}(n_{\mathrm{m}})$ from increasing by involving teachers which have their own requests into the operation, as described later in Section 5.2, though it creates additional user education and support costs and additional troubleshooting cost. The sum of additional user education and support costs and additional troubleshooting cost will get lower than the additional setting cost for changing policies for each application software, every time, as soon as teachers request for it.

3. Related Work

Computer-Based Testing (CBT) is a form of testing to measure the proficiency of students [13], [14]. Instead of papers, it uses computers for testing to improve overall efficiency. When a strict examination is held with computers, the use of application software that is not needed for the examination has to be restricted. For this purpose, a special system to prohibit the execution of unnecessary application software is sometimes provided by the examination authority.

This kind of system usually statically changes the rights of the execution of application software, while our method dynamically controls it. Any teacher during class can create strict examination environments in real time with the proposed method.

Application streaming is another method to create individual application software environments [15], [16], [17]. Images of application software can be distributed from a central server to educational PCs on demand with this method. If all of the application software used in an institution could be streamed, the administrators in the institution could theoretically create arbitrary application software environments on their educational PCs with one model disk image.

A major drawback of this method, currently, is system cost. A powerful central server is needed to stream application software. The license cost of streaming software itself is also high. The procedure to maintain application software images creates new operating cost. Moreover, in most systems, teachers, not administrators, cannot change application software environments dynamically during class. This method also does not control locally installed application software.

There are some commercial products to control the execution of installed application software on Windows PCs [18], [19], [20]. We, however, implemented a new system to show the feasibility of our method for some reasons. First, most of them were not originally made for the purpose of the reduction of the administrative burden with a disk image distribution system. Thus, to the best of our knowledge, none of them satisfied all of our demands which includes those of our future work. One system does not manage the number of occupied licenses and thus cannot restrict the execution of application software depending on the number of remaining licenses. Another system does not have a user interface for teachers to change application software environments dynamically. Moreover, we tried to reduce the system cost c_{s} in Eq. (1), by implementing the system without additional license fee. To ease future customization is also one of our reasons.

Group Policy, a feature of Windows, provides an infrastructure for centralized configuration management of the operating system [21]. When used in conjunction with Active Directory, Group Policy (Domain Group Policy) lets the administrators control the security and permissions for Organizational Units (OUs), certain groups of users or computers, within the domain [22]. As one of many features, Group Policy has a function to control the execution of application software (Software Restriction Policies) [23]. The centralized software restriction using this function in conjunction with Active Directory is a general countermeasure against malware in many business enterprises. Although Software Restriction Policies are changed in almost real time with the Group Policy Management Console (GPMC), it is not realistic to let teachers use it since it is a tool for administrators [24]. Note that our prototype utilizes Group Policy (Local Group Policy) as described later in Section 6.1.

4. Management of Educational PCs

This section presents an overview of the management of educational PCs using a disk image distribution system. Its problem is also clarified.

4.1 Management with Disk Image Distribution System

In order to let students use every educational PC in an institution in the same way, the administrators have to configure all of the educational PCs in the same state beforehand. If the administrators configured them individually, the management cost became high. A disk image distribution system is used in most institutions to address this issue. With the disk image distribution system, the administrators can distribute a disk image of one model educational PC to all of their educational PCs at once.

Depending on the timing of disk image distribution, there are two types of systems. One (former type) has to distribute a model disk image before actual use. The other (latter type) can distribute it when educational PCs are started. Since our university has a former system, we distribute a model disk image at midnight once a month.

Figure 1 shows a conceptual structure of disk image distribution systems. The *image distribution server* is introduced in the server room and stores model disk images. Each model disk image is distributed from the server to each educational PC in each PC room. Multicast distribution is usually deployed with the former system to distribute to groups of educational PCs.

4.2 Problem with Disk Image Distribution System

In higher educational institutions like universities, it is always required to create individual application software environments depending on requests from each department or each teacher.

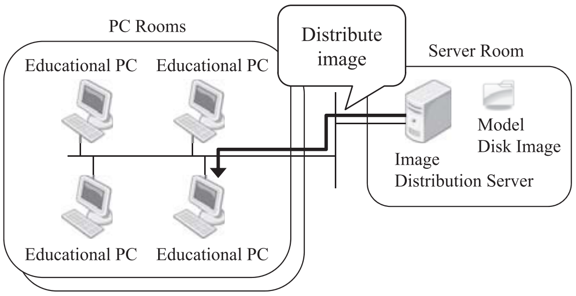


Fig. 1 A conceptual structure of the disk image distribution systems.

Each department has its own educational policies. Suppose that one department hopes to allow its students to use certain application software while the other departments have no interest in the application software. If no additional license fee is needed, the administrators can include the application software in the model disk image and distribute it to all of their educational PCs. However, if the same number of paid-for licenses as the number of installations is needed, it creates fruitless license cost. Essentially, it is reasonable to have the same number of licenses as the number of educational PCs in the PC room of the corresponding department in this case.

If the application software has floating license options, the license cost can be reduced [25]. However, if no sophisticated license servers are provided, it is difficult for students who actually need the application software to be allowed to use it. Students who do not belong to the corresponding department can waste a limited number of licenses. In addition, if the requested application software should be used after sufficient ethics and compliance education, the administrators will not want the other students to use it.

Note that, as described above, if a disk image distribution system is used with two or more images, the above problem is solvable to some extent [26]. Individual application software environments can be created by distributing a different disk image for the PC room in every department. This method, however, increases the number of model disk images which the administrators have to maintain. This leads to an increase in c_o in Eq. (1), as shown in Section 2. Moreover, if the target is changed into a subset of students in that department and the former type of image distribution system described in the previous subsection is used in that institution, this method cannot address the situation.

There is a way to allow teachers, or perhaps students, to create their own disk images and maintain them [27], [28], [29]. This method, however, is not universal since it requires the teachers, or the students, to have certain amount of skill and knowledge about computers. Instead of having flexibility, the cost to support them becomes high.

Furthermore, even when the latter type of the image distribution system described in the previous subsection is used, application software environments on the educational PCs are fixed after they are started. Teachers cannot dynamically change the environments during their classes.

5. Proposed Flexible Management Method with Execution Control

To address the problem described in the previous section, a flexible method to create individual application software environments, which does not rely on creating specific model disk images, is proposed in this paper. The proposed method controls the execution of application software on each educational PC in accordance with policies from departments or teachers. Teachers as well as administrators can directly and dynamically change its configuration during class.

A functional structure of the proposed method is shown in

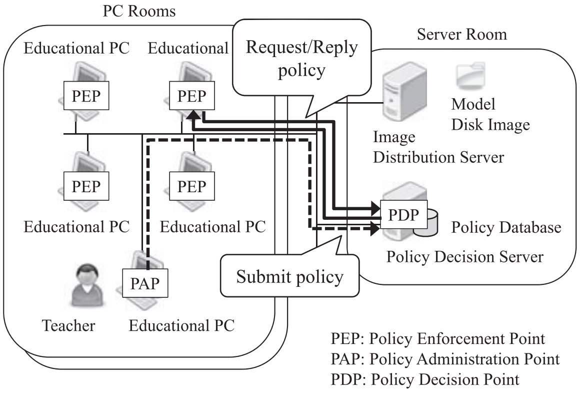


Fig. 2 A functional structure of the proposed method.

Fig. 2. The proposed method has three components. One is *Policy Enforcement Point* (*PEP*). Another is *Policy Administration Point* (*PAP*). The other is *Policy Decision Point* (*PDP*).

PEP is deployed on each educational PC. It actually controls the execution of application software in accordance with decided policies from its PDP. PAP submits changes in policies to its PDP on behalf of teachers. The teachers can modify their policies using their PAP. PDP on the *policy decision server* has a responsibility for policy decisions. It stores policies from its PAPs with its *policy database*. When a PEP requests policies for the corresponding educational PC or student, PDP searches its policy database, and replies with the result. PDP also actively notifies its PEPs of changes to its policy database.

Details of these components are described in the following subsections.

5.1 Policy Enforcement Point (PEP)

PEP is deployed on each educational PC to directly control the execution of application software in accordance with decided policies from its PDP. Since it is indispensable for the proposed method, it is configured to start automatically as a service when each educational PC is started. Students must not be allowed to terminate the service.

When starting up, each PEP requests initial policies (initial request) associated with the combination of corresponding user ID and IP address of its educational PC. After receiving a reply, the PEP enforces the initial policies on the educational PC. The received initial policies contain a *default policy* for the combination. It determines default rules for unspecified application software. Students have rights to execute all of the application software except for those which are expressly denied if retrieval of the default policy is set to *permit*. They cannot use all of the application software except for those which are expressly allowed if their default policy is set to *deny*.

After initial setup, the PEP waits for additional instructions from its PDP. If associated policies stored in the policy database are changed, the PEP receives the modifications from the PDP. The PEP then enforces the new policies on its educational PC. If the use of certain application software running on the educational PC is changed into prohibition, the PEP urges its student to quit it and finally forces it. The PEP also requests policies (on-demand request) to its PDP each time the student tries to use the application software whose simultaneous use is limited. Without obtaining permission, the PEP continues restricting the use of the corresponding application software.

As for the identification of certain application software, there are possibly two ways. One is based on the name or path of the corresponding executable file. The other is based on its hash. The proposed method employs the latter to prevent students from spoofing allowed application software. A file that is renamed or moved to other folders results in an identical hash [23]. Thus, students cannot avoid deny rules just by renaming or moving executable files.

Note that if the students could change unused bytes of these executable files, the students could avoid the deny rules. This problem could be solved by restricting the execution of all of the application software whose executable file does not have valid digital signatures. Our prototype, described later in Section 6, did not employ this method for the following reasons. First, there is a lot of application software whose executable file is not digitally-signed at present and thus it is hard to manage educational PCs based on this method. Next, according to the warning on their user interface, certificate rules in Software Restriction Policies, described later in Section 6.1, have a performance issue. Also, individual rules cannot be defined for multiple executable files digitally signed by the identical software vendor using certificate rules in the Software Restriction Policies. Finally, we thought that the risk was acceptable for the management of educational PCs as the attacks require a certain level of skills and many students will not attempt the attacks by taking risks.

5.2 Policy Administration Point (PAP)

PAP is used to configure policies in the policy database. Authorized teachers as well as the administrators can create individual application software environments at any time using their PAP. If a teacher wants to change the enforcement policies in his or her class, the teacher submits a new policy along with the specified user ID or IP address (or subnet address). We tried to prevent $c_{\mathrm{o,pro}}(1)$ and $c_{\mathrm{o,pro}}(n_{\mathrm{m}})$ in Eq. (4) from increasing, by allowing teachers themselves to directly configure their policies.

Since the subnet address for a specific classroom is always fixed, using subnet addresses is simple for normal classes. As for using user IDs, the current version of the proposed system assumes that teachers have the lists of user IDs of registered students for their classes in advance. For example, in our university, the teachers can get the lists of student IDs of registered students from the academic affairs system, and they can convert the lists into the lists of user IDs using another web-based tool. A way to cooperate with other systems to simplify this operation has to be discussed in our future work.

PAP also has functions to specify the number of simultaneous use for certain application software. Floating license options are supported with this function even when no special license servers are provided. Although there is a case where multiple applications (or multiple versions of application software) are associated with a single license (e.g., a software suite which consists of multiple applications), the proposed method does not currently support this case. To handle this, a sophisticated application grouping technique is needed, which we have started to address [30]. Details will be considered in our future work. PAP also has a function to register the default policy.

Moreover, as described in the previous subsection, the proposed method employs the hash of the executable file for application software to identify and control its execution. Teachers have to register the hash of the executable file for the application software before they can configure policies for the application software. Hence, PAP has another function to help the teachers calculate and register the hash.

When teachers log out, their PAP notifies its PDP of it to delete their policies from the policy database since it is considered that they finished their class.

5.3 Policy Decision Point (PDP)

PDP on the policy decision server stores policies in the policy database and has the responsibility for policy decisions. When a request from each PEP arrives, the PDP searches its policy database with the specified user ID and IP address, and replies with a result. The PDP checks the number of simultaneous use as well if it is limited for the associated application software.

Since PDP can be a single point of failure, special care for high availability is needed. Current virtualization and redundancy technologies can help this. Note that, if PDP consists of two servers for redundancy (e.g., an active PDP and a standby PDP), the policy databases on both servers have to be synchronized [31]. Since how to treat the single point of failure affects c_s in Eq. (1), a countermeasure has to be carefully selected according to the required service level.

PDP also has a function to notify its PEPs of changes in its policy database. If teachers add or delete their policy using their PAP, the PDP checks whether certain PEP is affected with the modification or not. First, the PDP creates the list of running PEPs associated with the added or deleted policy. Then, for each PEP on the list, the PDP searches its policy database again, and sends its result.

PDP stores policies from administrators and from teachers separately in its policy database. More precisely, it separately stores those policies for each default policy.

It first searches policies from administrators. As a resulting action, the administrators can specify one of the following four actions: *permit*, *deny*, *weak permit*, or *weak deny*. If permit or deny is selected in the matched policy, PDP replies with the resulting action immediately. Otherwise, PDP subsequently searches for policies from teachers. As a resulting action, teachers can specify either of the following two actions: *permit* or *deny*. If a matched policy is found in the policies from teachers, PDP replies with the resulting action. Otherwise, PDP replies with permit (or deny) when weak permit (or weak deny) was specified in the matched policy from administrators. PDP finally replies with permit (or deny) when the default policy is permit (or deny).

The proposed method employs this kind of hierarchy in order for the administrators not to lose definitive decisions. The introduction of weak permit and weak deny enables the administrators to delegate their authority to the limited scope.

A flowchart of this determining process including consideration of simultaneous use is summarized in **Fig. 3**. If the number of simultaneous use for certain application software is limited when an initial request arrives, PDP replies by tentatively denying to save finite resources. PDP repeats this process when policies for

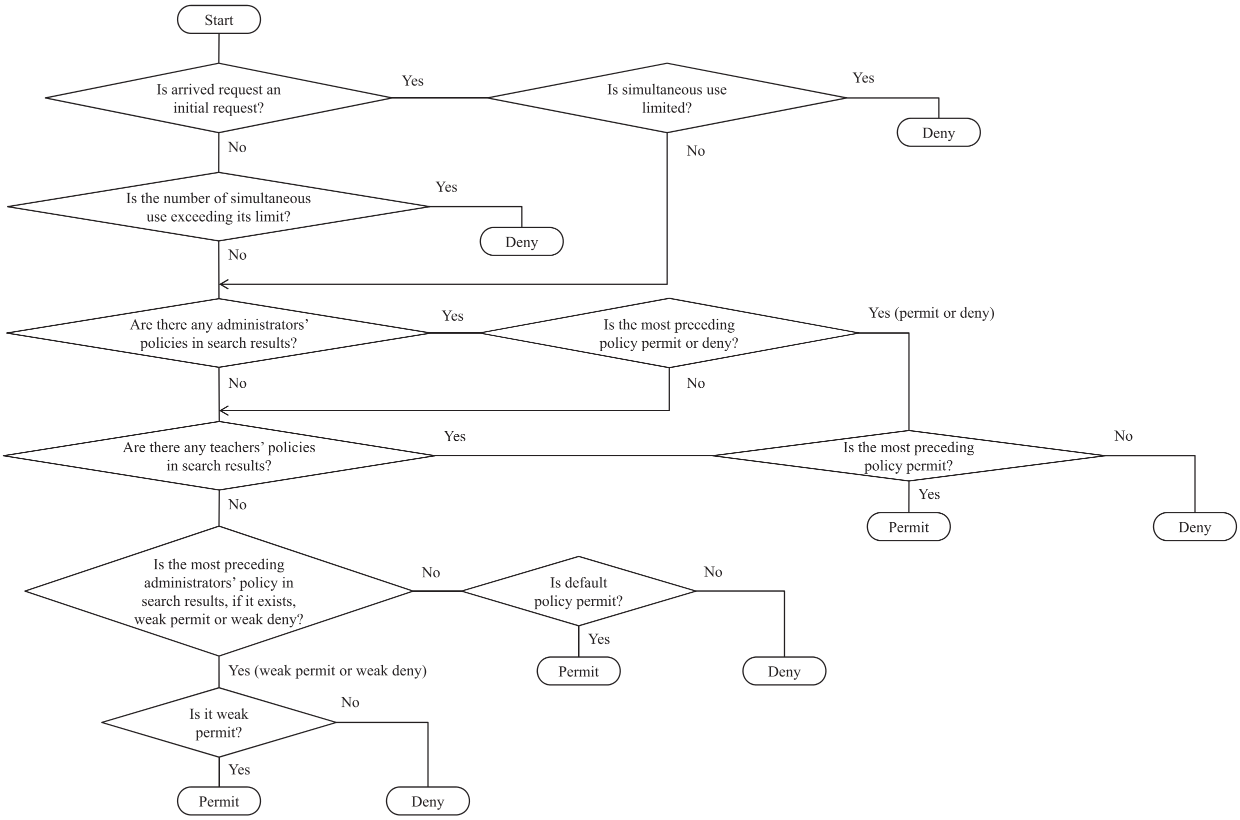

Start
Is arrived request an initial request?
Yes
Is simultaneous use limited?
Yes
Deny
No
No
Is the number of simultaneous use exceeding its limit?
Yes
Deny
No
Are there any administrators' policies in search results?
Yes
Is the most preceding policy permit or deny?
Yes (permit or deny)
No
No
Are there any teachers' policies in search results?
Yes
Is the most preceding policy permit?
No
Yes
Permit
Deny
No
Is the most preceding administrators' policy in search results, if it exists, weak permit or weak deny?
No
Is default policy permit?
No
Yes
Permit
Deny
Yes (weak permit or weak deny)
Is it weak permit?
No
Yes
Permit
Deny

Fig. 3 A flowchart for determining the result.

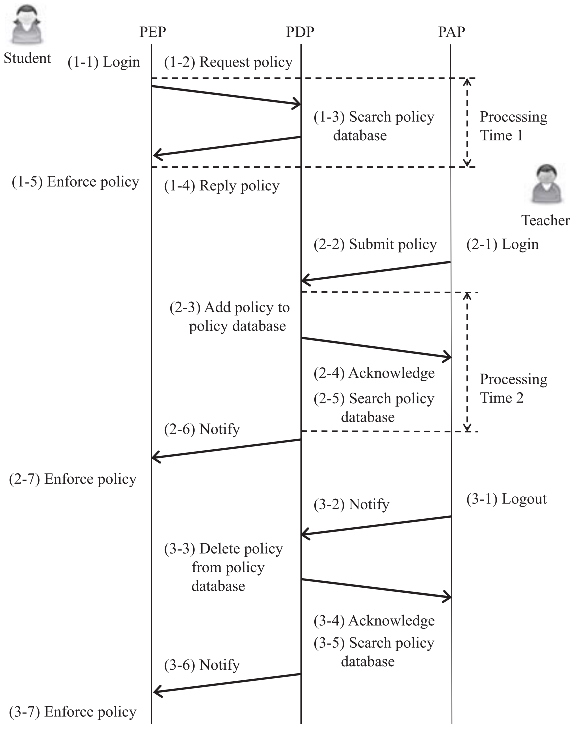


Fig. 4 An operation procedure of the proposed method.

multiple applications are requested (e.g., when receiving an initial request).

5.4 Operation Procedure

An operation procedure of the proposed method is shown in **Fig. 4**. It shows the procedure from before a student logs in to an educational PC to after a teacher logs out of the system. The teacher changes his or her policies during his or her class. Note that *Processing time 1* and *Processing time 2* are referenced later in Section 6.4. The operation procedure, consisting of three kinds of procedures, is depicted in Fig. 4 and can be summarized as follows.

First, the initial procedure after the students' login is described. When a student logs in to an educational PC (1-1), the PEP on the educational PC requests initial policies to its PDP by informing the PDP of the associated user ID and IP address (1-2). Receiving the request, the PDP searches its policy database (1-3) and replies with the initial policies for the requesting PEP (1-4). Receiving the reply, the PEP enforces the received initial policies (1-5).

Next, the policy configuration procedure during classes is described. When a teacher logs in to another educational PC and configures his or her policy (suppose that it is related to the student) using the PAP on his or her educational PC (2-1), the PAP submits the new policy to its PDP (2-2). Receiving the submission, the PDP adds the new policy to its policy database (2-3) and acknowledges the submission to the PAP (2-4). The PDP then searches its policy database to find related PEPs (2-5) and notifies the PEPs of the modification of its policies (2-6). Receiving the notification, each PEP enforces the received new policy (2-7).

Finally, the cleanup procedure after classes is described. When the teacher logs out of the system after his or her class finishes (3-1), the PAP notifies its PDP of it (3-2). Receiving the notification, the PDP deletes his or her policies from its policy database (3-3) and acknowledges the notification to the PAP (3-4). The PDP then searches its policy database to find related PEPs (3-5) and notifies the PEPs of the modification of its policies (3-6). Receiving the notification, each PEP enforces the modified policies (3-7).

6. Implementation

Execution Control Program, *Configuration Tool*, and *Policy Decision Server* were implemented as prototypes of PEP, PAP, and PDP, respectively. Since they were prototypes, they did not have all required features including precise error management features nor were not fully optimized for scalability yet. For example, no indexes were created on the Policy Database. Mutual authentication between those components was not implemented, though use of SSL authentication and encryption will be a candidate. To prevent $c_{\mathrm{o,pro}}(1)$ and $c_{\mathrm{o,pro}}(n_{\mathrm{m}})$ in Eq. (4) from increasing, it would be reasonable to use long-lived (half or all of the contract period of the educational system) certificates within relatively closed networks like networks for educational systems.

We conducted some basic operation tests of these prototypes and confirmed that they worked as we expected. We also evaluated the fundamental performance of the prototypes.

Details of the prototypes, basic operation tests, and fundamental performance evaluations are shown in the following subsections.

6.1 Execution Control Program

The Execution Control Program was implemented in C#.

First of all, a function to control the execution of certain application software was implemented using Software Restriction Policies in Group Policy, as described above in Section 3. More precisely, not those in Domain Group Policy for OUs, but those in Local Group Policy for local computer were utilized, since OUs are usually defined as relatively static units such as one's organization for manageability.

Software Restriction Policies have four types of rules to control the execution of application software: hash rules, certificate rules, path rules, and Internet zone rules [23]. Hash rules identify application software based on the hash of associated executable file. Certificate rules identify application software based on the signer of the digital signature for the associated executable file. Individual rules cannot be defined for multiple executable files digitally signed by the identical software vendor using certificate rules as described above in Section 5.1. Path rules identify application software based on the path of the associated executable file. Internet zone rules only apply to files with the .msi extension, which are Windows Installer packages.

Rules for specific application software are implemented with the execution control using hash rules in the Software Restriction Policies as designed above in Section 5.1. The execution control using hash rules in the Software Restriction Policies is performed when the following three data are written in the registry. One is *HashAlg* (for Hash Algorithm), another is *ItemData*, and the other is *ItemSize*. Since MD5 is used for the calculation of the hash, HashAlg is always 32,771 (in decimal). ItemData contains the hash of the associated executable file. ItemSize includes the size of the executable file. When receiving policies from its Policy Decision Server, each Execution Control Program adds these three data into the registry or deletes them from the registry.

Default deny policies are implemented using the execution controls using path rules in the Software Restriction Policies. An asterisk representing a wild-card character is written into the registry as a specified path. This means use of all of the application software is prohibited on that PC. Note that since hash rules have priority over path rules, the execution of application software that students are allowed to use is controllable using hash rules even under default deny policies.

A feature to urge and force students to quit specified application software was implemented as follows. First, each Execution Control Program extracts all of the processes during execution by using a class library of C#. Then it compares hashes of the executable files for these processes with the hash of the executable file for the specified application software. If a matched process is found, the Execution Control Program displays a message to urge a student to quit the application software. The student can preserve contents of the current work at this stage. After a definite period of time, the program issues a *close* command to alert the student that he or she does not have much time left. Nevertheless the student still uses the application software after another definite period of time. Then, the Execution Control Program issues a *kill* command to force the application software to quit.

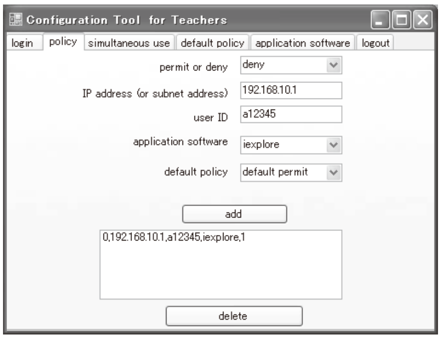
Moreover, each Execution Control Program has a user interface for students. Before using certain application software that the institution has a limited number of licenses for, a student has to push the button corresponding to the application software on his or her Execution Control Program to issue an on-demand request. This button also has other roles to show the student current policies and to let the student release the right of use when he or she finishes using the associated application software whose simultaneous use is limited. When the student logs out of the educational PC, the Execution Control Program notifies the Policy Decision Server of it using its closing procedure.

In actual environments, there are several cases when some error management features are required, such as when the interruption of power supply or network access for specific or all educational PCs occurs. In these cases, the Execution Control Program cannot complete its closing process to release the right of use for the application software whose simultaneous use is limited or cannot receive directions from the Policy Decision Server. Our current prototypes do not have any specialized countermeasures for these. This is a remaining implementation issue though timeout and retransmission mechanisms may address it.

6.2 Configuration Tool

The Configuration Tool was also implemented in C#.

A screenshot of the Configuration Tool is shown in **Fig. 5**. It has six tabs: *login tab*, *policy tab*, *simultaneous use tab*, *default policy tab*, *application software tab*, and *logout tab*. The login tab and logout tab are used for teachers to log in to and logout of their Configuration Tool, respectively. Currently, authentication and authorization mechanisms do not cooperate with the integrated authentication system of our university. This extension will be included in our future work. Well-used LDAP-based authentication and authorization will be candidate solutions.



permit or deny: deny

IP address (or subnet address): 192.168.10.1

user ID: a12345

application software: iexplore

default policy: default permit

0,192.168.10.1,a12345,iexplore,1

Fig. 5 A screenshot of the Configuration Tool.

Figure 5 shows a screenshot when the policy tab is selected. This tab is used for teachers to manage their current policies. After inputting its configuration parameters into each combo box and text box, a teacher pushes the *add button*. Then, the Configuration Tool submits the input policy to its Policy Decision Server. The Configuration tool also displays the input policy on the list box on the lower part. To delete a submitted policy, the teacher selects the corresponding policy and pushes the *delete button*. The current prototype has not provided a way to reorder the registered policies in arbitrary order. A new policy is simply added as the most prior policy. It also currently has no feature to apply the same policy to groups of students without using subnet addresses. These features will be included in our future work.

The simultaneous use tab is used for teachers to configure the number of allowed simultaneous use of the specific application software. After inputting its configuration parameters (the number of allowed simultaneous use, user ID, IP address (or subnet address), and target application software) into each combo box and text box, a teacher pushes the *add button* (it is on the simultaneous use tab and is not shown in Fig. 5). Then, the Configuration Tool submits the input policy of the simultaneous use to its Policy Decision Server. The Configuration Tool also displays the input policy of simultaneous use on the list box on the lower part. To delete a submitted policy of simultaneous use, the teacher selects the corresponding policy and pushes the *delete button* (it is also on the simultaneous use tab and is not shown in Fig. 5).

The default policy tab is used for teachers to change the default policy for each student. After inputting its configuration parameters (default policy, user ID, and IP address (or subnet address)) into each combo box and text box, a teacher pushes the *add button* (it is on the default policy tab and is not shown in Fig. 5). Then, the Configuration Tool submits the input default policy to its Policy Decision Server. The Configuration Tool also displays the input default policy on the list box on the lower part. To delete a submitted default policy, the teacher selects the corresponding policy and pushes the *delete button* (it is also on the default policy tab and is not shown in Fig. 5).

The application software tab is used for teachers to register the hash of the executable file for unregistered application software. The *open button* (it is on the application software tab and is not shown in Fig. 5) helps each teacher to input its configuration pa-

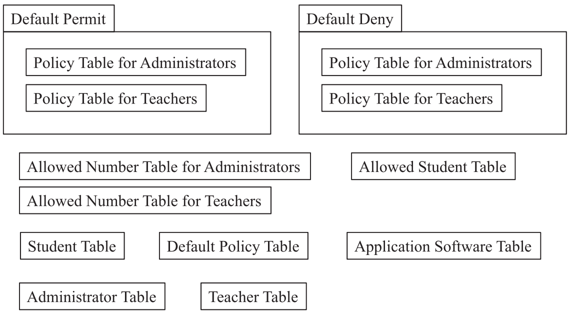

Default Permit
Policy Table for Administrators
Policy Table for Teachers
Default Deny
Policy Table for Administrators
Policy Table for Teachers
Allowed Number Table for Administrators
Allowed Student Table
Allowed Number Table for Teachers
Student Table
Default Policy Table
Application Software Table
Administrator Table
Teacher Table

Fig. 6 Tables in the Policy Database.

rameters: file name, file path, file hash, and file size. When the teacher pushes this button, the list of all of the executable files on his or her educational PC is displayed. All the teacher has to do at this point is to select the intended executable file from this list. Above four parameters are displayed automatically after the calculation of the hash of the executable file. The Configuration Tool submits these parameters when the teacher pushes the *submit button* (it is also on the application software tab and is not shown in Fig. 5).

6.3 Policy Decision Server

The Policy Decision Server was implemented in Perl. MySQL was selected as the DBMS for the Policy Database.

The following ten kinds of tables were created in the Policy Database: *Policy Table for Administrators*, *Policy Table for Teachers*, *Allowed Number Table for Administrators*, *Allowed Number Table for Teachers*, *Allowed Student Table*, *Student Table*, *Default Policy Table*, *Application Software Table*, *Administrator Table*, and *Teacher Table*. Policy Table for Administrators and Policy Table for Teachers exist for each default policy. A figure of the tables in the Policy Database is shown in **Fig. 6**.

The Policy Tables for Administrators and Policy Tables for Teachers are used to store policies from administrators and teachers, respectively. Each record in these tables has the application ID, resulting action (permit, deny, weak permit, and weak deny for administrators, and permit or deny for teachers), associated user ID and IP address (or subnet address), and registrant user ID. When a policy from a teacher or an administrator has arrived, the Policy Decision Server adds or deletes the associated record.

The Allowed Number Table for Administrators and Allowed Number Table for Teachers store the number of allowed simultaneous use. The limitation of simultaneous use is controlled based on these tables. Each record in those tables has application ID, the number of allowed simultaneous use, associated user ID and IP address (or subnet address), and registrant user ID. When a policy of simultaneous use from a teacher or an administrator has arrived, the Policy Decision Server adds or deletes the associated record.

The Allowed Student Table holds the application ID and user ID and IP address of all of the students who are currently allowed to use the corresponding application software. Further use of certain application software is prohibited when the number of records corresponding to the application software reaches the number of simultaneous use defined in the Allowed Number Tables. When the Policy Decision Server receives a notification of the release of the right of use for application software whose simultaneous use is limited, from an Execution Control Program, it deletes the corresponding record from the Allowed Student Table. It also deletes the corresponding records from the Allowed Student Table when it receives the notification of logout of a student.

The Student Table holds user ID and IP address of all of the students who currently log in to educational PCs. When a student logs in to an educational PC, its Execution Control Program sends an initial policy request. The Policy Decision Server obtains the user ID and IP address of the student from this request and adds it into the Student Table. When the Policy Decision Server receives the notification of logout of the student, it deletes the corresponding record from the Student Table.

The default action of students is specified in the Default Policy Table. Each record holds the default action, associated user ID and IP address (or subnet address), and registrant user ID.

The Application Software Table holds parameters to identify specific application software and control its execution on each Execution Control Program. Each record in this table has the application ID, file name, file path, file hash, and file size of the executable file for the corresponding application software.

The Administrator Table and Teacher Table hold the user ID and password of all of the administrators and all of the teachers, respectively. Each record in the Teacher Table also has the current IP address of an associated teacher.

First, when a request (an initial or on-demand request) from an Execution Control Program has arrived, the Policy Decision Server decides a response based on the contents of its Policy Tables and Allowed Number Tables. If the current default policy of the associated Execution Control Program represents permit, then the Policy Tables for the default permit is used. Otherwise, the Policy Decision Server refers to the Policy Tables for the default deny.

The Policy Decision Server determines the most appropriate policy for the requesting Execution Control Program by sequentially issuing SQL requests to these tables. For each table, it extracts matched policies using a *select* statement. As the subnet addresses are stored as simple decimal digit sequences with dots and a slash in the current prototypes, a Perl function checking whether an IP address (also stored as a simple decimal digit sequence with dots) is within a subnet address or not is called when the matched policies are described using subnet addresses.

Policy Tables for Administrators have priority over Policy Tables for Teachers in order for the administrators not to lose definitive controls of the execution of application software, as already described in Section 5.3. Administrators should not use permit policies along with deny all policies to describe user or device license options so that teachers can disallow the use of corresponding application software. The administrators should use weak permit or weak deny policies instead.

The Allowed Number Table for Administrators also has priority over Allowed Number Table for Teachers, though we suppose that it is quite rare that both an administrator and a teacher configure the number of allowed simultaneous use for the same

Table 1 An example of Policy Table for Administrators for default permit.

Application ID	Resulting action	IP address (subnet address)
1	Weak permit	192.0.2.0/24
1	Deny	0.0.0.0/0
2	Weak deny	192.0.2.0/24
2	Deny	0.0.0.0/0

Table 2 An example of Allowed Number Table for Administrators.

Application ID	Allowed number	IP address (subnet address)
1	50	192.0.2.0/24

application software.

Tables 1 and **2** show examples of a part of the Policy Table for Administrators for default permit and Allowed Number Table for Administrators, respectively. Use of application 1 is restricted only to a specific PC room and its simultaneous use is limited to fifty. Teachers in the PC room can still disallow the execution of the application in their classes. Use of application 2 is also restricted only to the specific PC room. Students are, however, not allowed to use the application even in the PC room without an explicit allowance from teachers.

Next, when a request for adding or deleting a policy from the Configuration Tool, the Policy Decision Server first specifies Execution Control Programs affected by the policy change. It utilizes the Student Table to create the list of affected Execution Control Programs. It calls the above Perl check function for each record in the Student Table to extract the affected Execution Control Programs when the added or deleted policy is described using a subnet address to specify its targets. The Policy Decision Server then reevaluates the policies for each affected Execution Control Program and sends the resulting policies to the Execution Control Program.

Moreover, when new managed application software is submitted from a teacher, the Policy Decision Server searches its Application Software Table. When the application software that has the identical hash does not already exist, the Policy Decision Server registers its application ID, file name, file path, file hash, and file size on its Application Software Table. The Policy Decision Server also sends these parameters to the Execution Control Programs and Configuration Tools currently running.

6.4 Basic Operation Tests and Fundamental Performance Evaluations

We prepared two PCs having the Execution Control Program as educational PCs for students, one PC having the Configuration Tool as an educational PC for teachers, and one Policy Decision Server within an experimental network. Some basic operation tests including the following were conducted and all of the fundamental features of the proposed method were confirmed to work as expected.

- Test for behavior against the initial request
- Test for behavior against the on-demand request
- Test for behavior against the change of policy
- Test for behavior against the change of managed application

Moreover, we evaluated the fundamental performance of the prototypes within the above experimental environments. To investigate scalability of the proposed method, two kinds of processing time of the Policy Decision Server were measured since a large number of multiple simultaneous requests or notifications were created within the associated processes, compared to the other processes.

One is the time from just start receiving an initial request (after the *accept* socket function and before the *recv* socket function in the source code of the Policy Decision Server program, which is omitted due to space limitations) and to just finish replying initial policies (after the *close* socket function in the source code of the Policy Decision Server program) [32]. It is the time to process the operations (1-2)–(1-4) in Fig. 4, as shown as Processing time 1 in Fig. 4.

We measured this processing time to evaluate the tolerance for simultaneous start of educational PCs. The parameters affecting this measurement contain the number of educational PCs, the number of rules on the Policy Tables and Allowed Number Tables, and the number of application software in the Application Software Table. Among these parameters, the numbers of rules and application software affect the database search time. The number of rules is affected by the numbers of specific groups for each application software, members of the groups, and PC rooms (subnet addresses) in turn.

As in a typical educational institution, this paper has the following presumptions. The number of application software whose execution has to be separately controlled is within a dozen or a few dozen at most. The number of specific groups for each application software is within a dozen. The number of members of the groups, such as the number of students taking a specific class, is within several dozen or a few hundred at most. The number of PC rooms is within a dozen or a few dozen at most. Recent database servers can select specific records from several tens of thousands of records in a short amount of time though it needs some performance tuning of the database. We thus chose the number of educational PCs as the only variable parameter in this paper. 50, 100, and 150 Execution Control Programs were emulated on one PC, respectively. These numbers translate to 5, 10, and 15 percent of the educational PCs in our university start almost simultaneously. Multiple initial requests were received almost simultaneously (within a few seconds) on the Policy Decision Server in the experiments.

Following the above supposition, we prepared five kinds of managed application software and ten records of policies for each application (fifty records in total) within the Policy Table for Teachers for default permit. The number of rules for each application was chosen based on the number of maximum PC rooms (subnet addresses) in our assumptions. This number may be a few thousand at most when user IDs are used for describing the rules. We let the Policy Table for Administrators for default permit and both Allowed Number Tables have no records, for simplicity. We had each record of policies within the Policy Table for Teachers specify the corresponding Execution Control Programs using only user IDs and only subnet addresses, respectively, and had no record have a policy related to the requests. That is, all five kinds of managed application software were allowed to use by the default policy.

We evaluated whether or not the processing times were small

Table 3 Evaluation results when all of the policies were described using only user IDs.

Operation	# of Execution	Processing time		
	Control Programs	Ave. [s]	Min. [s]	Max. [s]
(1-2)–(1-4)	50	0.425	0.235	0.508
	100	0.672	0.249	0.858
	150	0.748	0.258	1.02
(2-3)–(2-5)	1	0.002	0.002	0.003

enough against the time to finish starting educational PCs, that is about a few minutes.

The other is the time from just start adding a new policy and to just finish searching the Policy Database to find the Execution Control Programs affected by the modification. It is the time to process the operations (2-3)–(2-5) in Fig. 4, as shown as Processing time 2 in Fig. 4. We measured this processing time to evaluate the response time to modify the policies during classes. Since we have about a thousand educational PCs, we inserted a thousand records into the Student Table in advance to emulate the maximum number of login user. We prepared five kinds of managed application software including one corresponding to the newly added policy and ten records of policies for each application (fifty records in total) within the Policy Table for Teachers for default permit, following the above assumption. We let the Policy Table for Administrators for default permit and both Allowed Number Tables have no records, for simplicity.

We had each policy including newly added specify the corresponding Execution Control Programs using only user IDs and only subnet addresses, respectively. A policy related to only one of the Execution Control Program (the corresponding student was listed in the last record in the Student Table) was added using the Configuration Tool when these policies were described using only user IDs. A policy related to 50, 100, and 150 Execution Control Programs (the corresponding students were listed as the last 50, 100, and 150 records in the Student Table, respectively) was added, respectively, using the Configuration Tool when these policies were described using only subnet addresses. As it was difficult to prepare dozens of educational PCs for students within the experimental environments, actual transmissions of the notification message to each corresponding Execution Control Program (in operation (2-6) in Fig. 4) were omitted (by commenting out *connect* and *send* socket functions) in this measurement.

We evaluated whether or not the processing times were small enough against the time that teachers can wait during their classes, that is within about a dozen seconds.

The results of the evaluations are shown in **Tables 3** and **4**. Tables 3 and 4 show the results when all of the policies were described using only user IDs and only subnet addresses, respectively. They are averages, minimums, and maximums of fifty measurements. Note that, some of them are average of averages, average of minimums, and average of maximums of 50, 100, and 150 processing times, respectively. The specifications of the Policy Decision Server used for these evaluations are shown in **Table 5**. It is not a high-performance computer but an average desktop computer.

As shown in Tables 3 and 4, the processing time of operations (1-2)–(1-4) was small, compared to about a few minutes. Moreover, the processing time of operations (2-3)–(2-5) was also small, compared to about a dozen seconds. These results show that our prototype did not have significant scalability limitations within these fundamental performance evaluations though we have to further evaluate its performance with other remaining features, such as mutual authentication between the components of the proposed method.

Table 4 Evaluation results when all of the policies were described using only subnet addresses.

Operation	# of Execution	Processing time		
	Control Programs	Ave. [s]	Min. [s]	Max. [s]
(1-2)–(1-4)	50	0.802	0.592	0.902
	100	1.49	0.941	1.70
	150	1.84	0.963	2.21
(2-3)–(2-5)	50	0.130	0.008	0.252
	100	0.260	0.009	0.503
	150	0.380	0.009	0.747

Table 5 The specifications of the Policy Decision Server used for our experiments.

OS	Fedora 15
CPU	Intel® Core^{TM}2 Duo Processor E7500 (2.93 GHz)
Memory capacity	2 GB

7. Conclusions

In this paper, a flexible method to manage application software on educational Windows PCs has been proposed and its prototype was implemented.

First, the management cost of educational PCs was formulated and related work was discussed. Then, the traditional method using a disk image distribution system and its problem were described. The problem was that it robbed the administrators of the flexibility of management. It was hard for the administrators to create individual application software environments without significant efforts by the administrators.

To address this problem, the proposed method controls the execution of individual application software on each educational PC. It lets the administrators restrict use of application software to specific groups of users or educational PCs without relying on creating specific model disk images, which causes heavy burden on the administrators. Teachers as well as the administrators can directly and dynamically change its configuration to apply any policies at any time. The administrators do not lose definitive decisions about the execution control of application software in the proposed method.

In future work, we plan to conduct a field test to evaluate the performance of the proposed system more precisely and to find any administrative issues of the proposed method. Some already-described future work including an application grouping technique described in Section 5.2 will be considered. A method to detect changes in hashes of the executable files for managed application software and update the corresponding registered hashes automatically will also be established, which we also already started to address [30]. We will have to consider a way to alleviate the burden on teachers newly created with the proposed method as well.

References

[1] Bo, Y., Yingfang, L., Junsheng, L. and Jianhong, S.: The Impact of Computer Based Education on Computer Education, *Springer CCIS*, Vol.234, pp.1–9 (2011).

[2] Jianhong, S.: Solving Strategies Research for the Negative Impact of Computer Technology on Education, *Proc. ETCS 2010*, Vol.1, pp.671–674 (2010).

[3] Teramoto, T., Okada, T. and Kawata, S.: A Distributed Education-Support PSE System, *Proc. e-Science 2007*, pp.516–520 (2007).

[4] Schaffer, H.E., Averitt, S.F., Hoit, M.I., Peeler, A., Sills, E.D. and Vouk, M.A.: NCSU's Virtual Computing Lab: A Cloud Computing Solution, *Computer*, Vol.42, No.7, pp.94–97 (2009).

[5] Masuda, H.: The Large Scale Educational Computer Systems, *IPSJ Magazine*, Vol.45, No.3, pp.225–226 (2004).

[6] Fujiwara, M., Kawano, K. and Yamai, N.: An Execution Control System for Application Software Reducing Administrative Burden of Educational PCs, *Proc. C3NET 2012*, pp.375–380 (2012).

[7] Fujiwara, M., Kawakami, T., Kawano, K. and Yamai, N.: On-demand Configuration Feature of Target Programs Manageable for Teachers on Application Execution Control System, *Proc. IOTS 2011*, Vol.2011, pp.59–66 (2011).

[8] Kawakami, T., Kawano, K. and Yamai, N.: Teacher Configurable Execution Control System for Application Software on Educational Windows PCs, *Proc. IOTS 2010*, Vol.2010, pp.1–8 (2010).

[9] Microsoft Windows, Microsoft (online), available from ⟨http://windows.microsoft.com/en-us/windows/home⟩ (accessed 2013-09-26).

[10] OS X Mountain Lion: Apple (online), available from ⟨http://www.apple.com/osx/⟩ (accessed 2013-09-26).

[11] The Community ENTerprise Operating System: CentOS Project (online), available from ⟨http://www.centos.org/⟩ (accessed 2013-09-26).

[12] The world's most popular free OS | Ubuntu: Canonical (online), available from ⟨http://www.ubuntu.com/⟩ (accessed 2013-09-26).

[13] Akdemir, O. and Oguz, A.: Computer-Based Testing: An Alternative for the Assessment of Turkish Undergraduate Students, *Computers & Education*, Vol.51, No.3, pp.1198–1204 (2008).

[14] Papanastasiou, E.C.: Computer-Adaptive Testing in Science Education, *Proc. CBLIS 2003*, pp.965–971 (2003).

[15] Application Virtualization: Microsoft (online), available from ⟨http://technet.microsoft.com/en-us/appvirtualization/⟩ (accessed 2013-05-27).

[16] XenApp: Citrix (online), available from ⟨http://www.citrix.com/products/xenapp/overview.html⟩ (accessed 2013-05-27).

[17] VMware ThinApp: VMware (online), available from ⟨http://www.vmware.com/products/thinapp/overview.html⟩ (accessed 2013-05-27).

[18] Kaspersky Endpoint Security for Business Select: Kaspersky (online), available from ⟨http://usa.kaspersky.com/business-security/endpoint-select⟩ (accessed 2013-05-27).

[19] McAfee Application Control: McAfee (online), available from ⟨http://www.mcafee.com/us/products/application-control.aspx⟩ (accessed 2013-05-27).

[20] Wingnet: Computer Wing (online), available from ⟨http://www.cwg.co.jp/?page_id=141⟩ (accessed 2013-05-27).

[21] Group Policy: Microsoft (online), available from ⟨http://technet.microsoft.com/en-us/library/cc725828(WS.10).aspx⟩ (accessed 2013-05-27).

[22] What is the difference between a domain, a workgroup, and a homegroup?: Microsoft (online), available from ⟨http://windows.microsoft.com/en-us/windows7/what-is-the-difference-between-a-domain-a-workgroup-and-a-homegroup⟩ (accessed 2013-09-19).

[23] Software restriction policies overview: Microsoft (online), available from ⟨http://technet.microsoft.com/en-us/library/cc759106(v=ws.10).aspx⟩ (accessed 2013-09-19).

[24] Group Policy management for IT pros: Microsoft (online), available from ⟨http://windows.microsoft.com/en-us/windows7/group-policy-management-for-it-pros⟩ (accessed 2013-09-19).

[25] Jian-ping, C. and Li-ping, Q.: Research and Application of the Floating License Management Strategy, *Proc. ICISE 2009*, pp.1797–1800 (2009).

[26] Burd, S.D., Gaillard, G., Rooney, E. and Seazzu, A.F.: Virtual Computing Laboratories Using VMware Lab Manager, *Proc. HICSS 2011*, pp.1–9 (2011).

[27] Rindos, A., Vouk, M., Vandenberg, A., Pitt, S., Harris, R., Gendron, D. and Danford, T.: The Transformation of Education through State Education Clouds, *IBM Global Education* (2010).

[28] Li, P.: Provisioning Virtualized Datacenters through Virtual Computing Lab, *Proc. FIE 2010, T3C*, pp.1–6 (2010).

[29] Vouk, M., Averitt, S., Bugaev, M., Kurth, A., Peeler, A., Shaffer, H., Sills, E., Stein, S. and Thompson, J.: "Powered by VCL" - Using Virtual Computing Laboratory (VCL) Technology to Power Cloud Computing, *Proc. ICVCI 2008*, pp.1–10 (2008).

[30] Okamoto, D., Fujiwara, M., Kawano, K. and Yamai, N.: Target Application Grouping Function Considering Software Updates on Application Execution Control System, *Proc. ADMNET 2013*, pp.627–632 (2013).

[31] MySQL 5.6 Reference Manual: 16 Replication, Oracle (online), available from ⟨http://dev.mysql.com/doc/refman/5.6/en/replication.html⟩ (accessed 2013-05-27).

[32] Sockets: Client/Server Communication, perldoc.perl.org (online), available from ⟨http://perldoc.perl.org/perlipc.html#Sockets%3a-Client%2fServer-Communication⟩ (accessed 2013-09-22).

Keita Kawano received his B.E., M.E., and Ph.D. degrees from Osaka University, Osaka, Japan, in 2000, 2002, and 2004, respectively. From October 2004 to March 2010, he was an Assistant Professor of the Information Technology Center, Okayama University, Okayama, Japan. From April 2010 to March 2011, he was an Assistant Professor of the Center for Information Technology and Management, Okayama University. Since April 2011, he has been an Associate Professor of the same center. His research interests include mobile communication networks and distributed systems. He is a member of IEEE and IEICE.

Daisuke Okamoto received his B.E. degree in engineering from Okayama University, Japan, in 2012. He is currently a master's student at the Graduate School of Natural Science and Technology, Okayama University. His research interests include distributed systems.

Masanori Fujiwara received his B.E. and M.E. degrees in engineering from Okayama University, Japan, in 2011 and 2013, respectively. He is currently with JR WEST IT Solutions Company. His research interests include distributed systems.

Nariyoshi Yamai received his B.E. and M.E. degrees in electronic engineering and his Ph.D. degree in information and computer science from Osaka University, Osaka, Japan, in 1984, 1986 and 1993, respectively. In April 1988, he joined the Department of Information Engineering, Nara National College of Technology, as a Research Associate. From April 1990 to March 1994, he was an Assistant Professor in the same department. In April 1994, he joined the Education Center for Information Processing, Osaka University, as a Research Associate. In April 1995, he joined the Computation Center, Osaka University, as an Assistant Professor. From November 1997 to March 2006, he joined the Computer Center, Okayama University, as an Associate Professor. Since April 2006, he has been a Professor in Information Technology Center (at present, Center for Information Technology and Management), Okayama University. His research interests include distributed systems, network architecture and the Internet. He is a member of IEICE and IEEE.