International Journal of Informatics Society

09/15 Vol. 7 No.2 ISSN 1883-4566

Informatics Society

Aims and Scope

The purpose of this journal is to provide an open forum to publish high quality research papers in the areas of informatics and related fields to promote the exchange of research ideas, experiences and results.

Informatics is the systematic study of Information and the application of research methods to study Information systems and services. It deals primarily with human aspects of information, such as its quality and value as a resource. Informatics also referred to as Information science, studies the structure, algorithms, behavior, and interactions of natural and artificial systems that store, process, access and communicate information. It also develops its own conceptual and theoretical foundations and utilizes foundations developed in other fields. The advent of computers, its ubiquity and ease to use has led to the study of informatics that has computational, cognitive and social aspects, including study of the social impact of information technologies.

The characteristic of informatics' context is amalgamation of technologies. For creating an informatics product, it is necessary to integrate many technologies, such as mathematics, linguistics, engineering and other emerging new fields.

Guest Editor's Message

Ryozo Kiyohara

Guest Editor of Twentieth Issue of International Journal of Informatics Society

We are delighted to have the Twentieth issue of the International Journal of Informatics Society (IJIS) published. This issue includes selected papers from the Eighth International Workshop on Informatics (IWIN2014), which was held at Prague, Czech Republic, Sep. 10-12, 2014. The workshop was the eighth event for the Informatics Society, and was intended to bring together researchers and practitioners to share and exchange their experiences, discuss challenges and present original ideas in all aspects of informatics and computer networks. In the workshop 24 papers were presented in five technical sessions. The workshop was successfully finished with precious experiences provided to the participants. It highlighted the latest research results in the area of networking, business systems, education systems, design methodology, groupware and social systems.

Each paper submitted IWIN2014 was reviewed in terms of technical content, scientific rigor, novelty, originality and quality of presentation by at least two reviewers. Through those reviews 15 papers were selected for publication candidates of IJIS Journal, and they were further reviewed as a Journal paper. This volume includes four papers among the accepted papers, which have been improved through the workshop discussion and the reviewers' comments.

We publish the journal in print as well as in an electronic form over the Internet. We hope that the issue would be of interest to many researchers as well as engineers and practitioners over the world.

Ryozo Kiyohara is received B.E., and M.E. degrees from Osaka University in 1983, and 1985. Since joining Mitsubishi Electric Corporation in 1985, he had been engaged in developing a machine translation system. From 1989, he had been at the Institute for the New Generation Computing Technologies engaged in the Fifth Generation Computing Project until he returned to Mitsubishi Electric Corporation in 1992. He received Ph.D. in 2008 from Osaka University in Information Science and Technology. He is a professor of Kanagawa Institute of Technology since 2012. His current research interests include software upgrade systems and Java processing in on-vehicle information devices. He is a member of IEEE (Senior Member), ACM, IPSJ (Information Processing Society of Japan), and IEICE (The Institute of Electronics, Information and Communication Engineers).

A Method for Detection of Traffic Conditions in an Oncoming Lane Using an In-vehicle Camera

Ryo Shindo*, and Yoh Shiraishi**

* Graduate School of Systems Information Science, Future University Hakodate, Japan
** School of Systems Information Science, Future University Hakodate, Japan
{ g2113016, siraisi }@fun.ac.jp

Abstract - In recent years, we have become able to acquire traffic information about traffic congestion through the VICS (Vehicle Information and Communication System). The VICS is one of the traffic systems that provide drivers with information on the state of traffic congestion. However, it is difficult for drivers to decide appropriately as to whether they should change lanes or make detours because the VICS provides information on the causes of traffic congestion, such as traffic accidents or road works, in the form of icons. Icons are simple representations, but are not intuitive and informative. In contrast, presenting images recorded by an in-vehicle camera to represent the causes of traffic congestion is more effective than presenting icons to help users to understand the causes intuitively. When an in-vehicle camera records the conditions directly in front of a moving vehicle, recording the traffic conditions of an oncoming lane is simpler than trying to record the conditions in the lane in which the user is driving (driving lane), as preceding vehicles may obscure the camera view. If images representing the conditions in front of preceding vehicles are sent to drivers from vehicles in the opposite lane in advance, the drivers can avoid the congestion effectively. Therefore, we propose a method for detecting the traffic conditions of an oncoming lane using an in-vehicle camera. In addition, we conducted some experiments to show the effectiveness of the proposed system. In particular, we conducted the experiment about estimating the speed of vehicles on an oncoming lane by using optical flow toward detecting the traffic congestion in an oncoming lane. The experimental results suggest that the length of optical flows changes depending on the speed of oncoming vehicles and the proposed method has potential to detect traffic conditions.

Keywords: in-vehicle camera, detection of vehicles, traffic congestion, sensing, estimation of vehicle speed

1 INTRODUCTION

Drivers cannot effectively avoid traffic congestion through methods such as changing lanes and making detours if they are not aware of conditions of traffic congestion, such as the causes and ranges of the congestion, in advance. The VICS (Vehicle Information and Communication System) is one of the traffic systems that provide information on the conditions of traffic congestion [1]. In the VICS, information such as the volume of traffic, the speed of vehicles, and so on is acquired by sensors located on roads and sent to the information center. The collected information is converted into traffic information. The center sends the traffic information to car navigation systems and other in-vehicle devices. However, the VICS provide information on the causes of traffic congestion, such as traffic accidents or under construction, in the form of icons. Icons are simple representations, but are not intuitive and informative for grasping traffic congestion. Therefore, it is difficult for drivers to decide how to avoid traffic congestion effectively.

Currently, Probe Information Systems are in wide-spread usage [2]-[4]. Probe Information Systems are systems that support aspects of driving, such as navigating and calling for attention, by using information collected by sensors embedded in vehicles. Probe information includes vehicles' location information, air temperature, engine rotation speed, actuating information of the ABS (Antilock Brake System), and so on. The collected probe information can be shared among vehicles through a network or directly with a wireless connection called "inter-vehicle communication" [5]-[7].

A driver's front view is partially obscured by the preceding vehicles in the driving lane when the driver tries to record the causes of the congestion using an in-vehicle camera. Consequently, the driver cannot grasp the causes of traffic congestion and cannot avoid traffic congestion in advance unless the driver comes close to the site of the cause. For example, in Fig. 1, the cause is in front of vehicle C. Vehicle A's front view is partially obscured by the preceding vehicles in the driving lane. The driver of vehicle A cannot grasp the causes of traffic congestion unless the driver comes at points of vehicle C. On the other hand, vehicles in the oncoming lane (oncoming vehicles), as shown vehicle B in this figure, can grasp the causes of traffic congestion in the driving lane. The driver of vehicle A can identify congestion in front of the preceding vehicles and avoid it if the driver gets images representing the causes of traffic congestion in his or her driving lane from oncoming vehicles in advance. In this figure, vehicle B can grasp the causes of traffic congestion in the opposite lane, and vehicle A can acquire an image representing the causes from vehicle B when vehicle B comes at the point of vehicle B*.

For these reasons, in this study we assume that vehicles can share images and we propose a method for detecting traffic congestion in an oncoming lane, by using an in-vehicle camera. This study aims to detect traffic congestion in an oncoming lane from the view point of vehicle B in this figure.

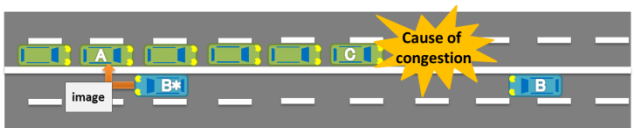


Figure 1: The positional relation between vehicles

This paper is organized as follows. Section 2 mentions research related to our study. Section 3 discusses the requirements of the proposed system. We outline our proposed method in Section 4. Finally, we discuss the effectiveness of our proposed method in Section 5.

2 RELATED WORK

This section introduces research related to our study. First, we discuss research and technologies related to presenting and sharing information on traffic conditions in Section 2.1. In addition, we discuss research on sharing information on traffic conditions by using an in-vehicle camera in Section 2.2. Finally, we discuss and compare the related research and our proposed method in Section 2.3.

2.1 Presenting and Sharing Information on Traffic Conditions

The VICS is one of the traffic systems that provide information on the conditions of traffic congestion [1]. In the VICS, information is collected by sensors located on roads and sent to information center. The collected information is converted into traffic information, such as the range of traffic congestion, road obstacles and highway regulations. The center sends the traffic information to car navigation systems and other in-vehicle devices using microwaves in the ISM band and frequency modulation (FM), similar to the Radio Data System (RDS) or Data Radio Channel (DARC). Thus the VICS can provide traffic information in real time. In the VICS, information displayed on maps of car navigation systems presents the traffic congestion classified into three degrees (sparse, crowded, and congested) based on the VICS's classification of traffic congestion (Table 1). VICS also displays icons representing highway regulations, hazard to moving traffic, and so on (Fig. 2). Drivers can grasp the traffic conditions anywhere by observing the displayed information.

However, the VICS cannot necessarily collect and provide this information for every road, because some roads do not have devices to collect information. In addition, the VICS provide information on the causes of traffic congestion, such as traffic accidents or under construction, as icons. Therefore, drivers must understand the meanings of the icons. However, drivers cannot decide whether or not they will avoid traffic congestion effectively because it is difficult for them to imagine the scale and the influence of the event that is happening in the driving lane from icons. Icons provided by the VICS are not intuitive information for drivers because they are simple information that does not depend on the scale of the causes.

Presenting camera images representing the causes of traffic congestion is effective for intuitive comprehension of traffic conditions [8, 9]. Intuitive comprehension enables drivers to identify traffic congestion in front of preceding vehicles and to avoid it in advance.

Figure 2: Icons provided by the VICS [1]

Table 1: The VICS's classification of traffic congestion [1]

Degree of congestion (Color)	General road	Inner-city high-speed way	Intercity high-speed way
Congested (Red)	Less than 10km/h	Less than 20km/h	Less than 40km/
Crowded (Orange)	10km/h-20km/h	20km/h-40km/h	40km/h-60km/h
Sparse (Green)	More than 20km/h	More than 40km/h	More than 60km/h

Tamai et al. [8] proposed a system that provides videos recorded at the point of traffic congestion for drivers' intuitive comprehension. A smartphone placed on the dashboard with a cradle records traffic congestion. The system collects and provides the recorded videos effectively, considering the time difference and the degree of congestion in the videos. The time difference means the difference between the time at witch a user receives the video and the time when the video was recorded. Tamai et al. [9] proposed a method that shares short videos representing the traffic conditions on roads with other vehicles. The system grasps the speed of a moving vehicle and determines the ranges of congestion based on the speed. The speed can be calculated based on location information acquired by a GPS sensor embedded in a smartphone placed on the vehicle's dashboard. At the same time, the smartphone records a front view. The system manipulates the video images considering the colors and the shapes, and detects traffic lights when the vehicle is in congestion. In addition, the system generates a video that is about 10 seconds long. The system grasps the speed of the moving vehicle easily by calculating the movement of traffic lights in the video because traffic lights are stationary objects.

2.2 Grasp of Traffic Conditions by Using In-vehicle Cameras

We will now introduce some research on grasping the conditions of roads by using an in-vehicle camera. Kutoku et al. [10] proposed a system that detects obstacles on roads by using an in-vehicle camera. An in-vehicle camera is placed on the dashboard of a moving vehicle and records the view in front of the moving vehicle. The system generates subtracted images by using the video currently being recorded and background video. Background video is a video recorded in advance on the same road when it had no obstacles. The system detects obstacles by using subtracted images. Many researchers tackle the detection of objects on roads. However, the objects targeted by such research are assumed objects such as a person, a vehicle, and so on. Kutoku's system can detect unexpected objects by using the

subtracted images. To generate subtracted images, the system must examine the time and position of the vehicles in the two videos because the speed and the positions of moving vehicles are different in each video. First, the system considers the time between the two videos using the scale representing the distance between the cameras in the two videos. Second, the system considers the positions of moving vehicles in each video by image processing of the surface of roads. According to this processing, the frames between two videos are selected and subtracted images are generated. The system calculates the recall, the false detection rate and the rate of false detection frames based on the distance between the moving vehicle and obstacles, by using image features of subtracted images. Image features include the brightness, the intensity and the edge. Then, the system detects unexpected objects considering the calculation results.

Hamao et al. [11] proposed a system that detects traffic congestion by using an in-vehicle camera. A smartphone is placed on a moving vehicle and records the view in front of the moving vehicle. The system sets a region of interest (ROI) on images, and calculates the standard deviation of the luminance histogram of the oncoming lane in the ROI. The system detects congestion based on the calculated standard deviation of the luminance histogram between congested roads and uncongested roads.

2.3 Comparing the Related Works with Our Method

Providing information on the conditions of traffic congestion using the VICS is not intuitive for drivers because the VICS presents such information as icons. The method proposed by Tamai et al. demonstrates that presenting information on traffic congestion as camera images taken by an in-vehicle camera is effective. However, in the case where preceding vehicles are moving in front of the vehicle with an in-vehicle camera, the camera cannot record the state of traffic congestion and its causes in the area in front of the preceding vehicles. Therefore, recording traffic congestion from an oncoming lane is easier than from a driving lane. To grasp the causes of the congestion by using an in-vehicle camera, it is necessary to detect the congestion and its range. In addition, to grasp the range and detect the congestion, it is necessary to detect the speed of oncoming vehicles. Grasping the ranges of the congestion and detecting the congestion are possible by acquiring the speed from oncoming vehicles with inter-vehicle communication. However, the moving vehicle must acquire the speed from a number of oncoming vehicles. On the other hand, detecting congestion is possible with only one moving vehicle with an in-vehicle camera. The method proposed by Kutoku et al. that detects road obstacles can detect congestion, but has difficulty detecting the speed of oncoming vehicles. The method proposed by Hamao et al. cannot detect the speed of oncoming vehicles. In addition, this method cannot discriminate between oncoming vehicles and objects behind them in images.

3 REQUIREMENTS OF THE PROPOSED SYSTEM

For intuitive grasping of the conditions of traffic congestion, presenting camera images is more effective than presenting icons. In addition, recording the conditions of oncoming lanes is easier than recording that of driving lanes when an in-vehicle camera records the view in front of a moving vehicle. Grasping the ranges of the congestion is required in order to detect the causes of the congestion. Grasping the ranges of the congestion is, namely, detecting the beginning and ending point of the congestion. Moreover, the speed of oncoming vehicles is required in order to grasp the ranges. In an image, oncoming vehicles and background objects behind them must be distinguished between when image processing is applied to the image. In this study, optical flows generated between two images are calculated in order to grasp the speed of oncoming vehicles. The optical flow is a line that represents the movement of objects between two images as a vector. The length of optical flow (LOF) generated from oncoming vehicles is calculated, and the speed of oncoming vehicles is calculated based on the length. In this way, the congestion is detected. LOF depends on the distance between a moving vehicle with an in-vehicle camera and oncoming vehicles, and the relative speed between the vehicles. The distance between the moving vehicle and oncoming vehicles is smaller than the distance between the moving vehicle and the objects behind oncoming vehicles. The movement of oncoming vehicles per a unit of time is different from that of the objects behind oncoming vehicles. In this way, oncoming vehicles and objects behind them are distinguished. In addition, LOF changes depending on not only the change in the speed of a moving vehicle but also the speed of oncoming vehicles. The speed of the moving vehicle can be calculated by using location information acquired by the GPS sensor embedded in the driving recorder and the smartphone.

Therefore, the speed of oncoming vehicles can be estimated by calculating the speed of the moving vehicle and the optical flows on the images from the in-vehicle camera. In addition, traffic congestion can be detected and images representing the causes of traffic congestion can be generated.

4 PROPOSED METHOD

4.1 Summary of the Proposed System

On the basis of the considerations as mentioned above, we propose a system to solve these problems. Figure 3 shows the positional relation of a moving vehicle, oncoming vehicles, and a cause of traffic congestion.

The proposed system needs to perform the following functions.

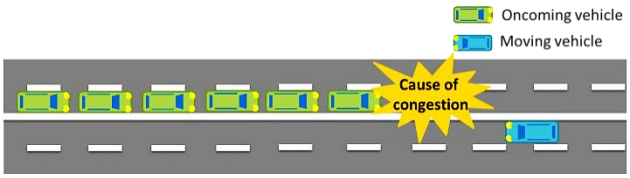


Figure 3: The positional relation between vehicles and the cause of congestion

A) Detect vehicles in an oncoming lane
B) Estimate the speed of oncoming vehicles
C) Detect traffic congestion
D) Find images representing the causes of traffic congestion
E) Estimate the range of traffic congestion

Figure 4 shows an overview of the proposed system.

First, a driver mounts a smartphone on the dashboard and the smartphone records the front view of an oncoming lane. At the same time, the speed of the moving vehicle is acquired by a GPS sensor. Second, the system generates the optical flows between two images recorded by the smartphone. In addition, the system calculates the LOF of each relative speed and stores the dataset of LOF and the relative speed in the Optical Flow Length Database (Optical Flow Length DB). Third, the system defines an interpolation function by using the dataset in the database to calculate the relative speed from LOFs that are not stored in the database. Fourth, the system estimates the speed of oncoming vehicles by using the newly calculated LOF and the function. The system decides that congestion is occurring in an oncoming lane if the estimated speed falls below the specified threshold. At the same time, the system generates an image representing the cause of the congestion by searching for an image recorded at the beginning of the congestion. Finally, the system generates the range of the congestion by using location information from the beginning and ending point of the congestion, and presents the image and the range on a map application.

4.2 The Way to Calculate Optical Flows

In this section, we explain how to calculate optical flows of oncoming vehicles in in-vehicle camera images. There are two general ways to calculate optical flows called Phase Correlation and Block Matching Method [12, 13]. Phase Correlation is a method that calculates optical flows using a contrast equation of luminance gradient with constraint conditions. Phase Correlation can calculate optical flows, but it makes errors and is especially affected by rapid luminance changes. Block Matching Method is a method that uses a particular part of an image as a template, and calculates optical flows by exploring the parts that fit the template in the next time image. It can calculate optical flows steadily, but it is more computationally expensive than Phase Correlation. In addition, Block Matching Method depends on the size and the features of the block in an image when optical flows are calculated considering the rotation and scaling of an image.

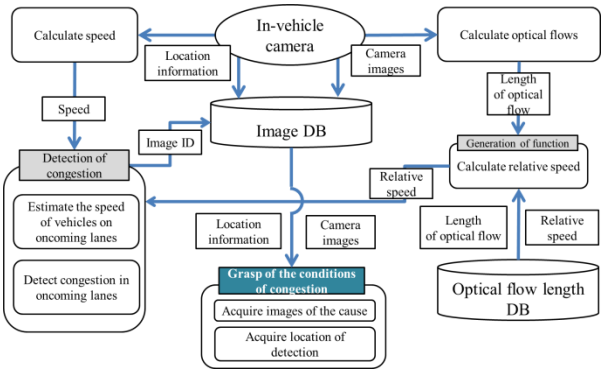


Figure 4: An overview of the proposed system

Figure 5: Optical flows drawn by LK method

In this study, vehicles and other objects in an oncoming lane are enlarged in the image because they are recorded by a moving vehicle on an opposite lane. Therefore, in this study, Block Matching Method is not appropriate to calculate optical flows. Our system uses the LK (Lucas-Kanade) method that is classified into Phase Correlation and calculates optical flows by detecting feature points of an image in order to reduce the errors of rapid luminance changes (Fig. 5).

However, in outdoor environment, it is difficult to diminish all noises caused by rapid luminance changes even with the use of the LK method. So the extraordinary optical flows are generated by these noises. Therefore, the proposed method sets thresholds for the length and the angle of the flows, and diminishes the flows that are out of the ranges decided by the thresholds. As a result, the extraordinary flows caused by the false detection of feature points are diminished.

4.3 Grasping the Conditions of Congestion

In order to grasp the conditions of congestion, the system uses two databases. One is an Image Database (Image DB) that stores recorded images and location information, and the other is an Optical Flow Length Database (Optical Flow Length DB) that stores LOF and the corresponding relative speed. Optical Flow Length DB is used to detect oncoming vehicles and to estimate the speed of oncoming vehicles. Table 2 and Table 3 show the structure of each database.

4.3.1 Detecting Vehicles , Estimating the Speed of Oncoming Vehicles

When optical flows are generated from objects in an oncoming lane in images, the flows generated outside the zone of an oncoming lane are unnecessary. Therefore, we define a region of interest (ROI) so that the oncoming

Table 2: The table structure of Image DB

Attribute Name	Detail
ID	Identification number of images
Image	Recorded image
Lat	Latitude of recording location
Lon	Longitude of recording location

Table 3: The table structure of Optical Flow Length DB

Attribute Name	Detail
R_{speed}	Relative speed between a moving vehicle and an oncoming vehicle
Len	LOF generated from oncoming vehicles

vehicles fit into the region in an image (Fig. 6), and optical flows are generated from the objects in the ROI.

The speed of the moving vehicle is calculated by using location information acquired by a GPS sensor of a smartphone when a driver drives the vehicle. We define the value representing the speed of the moving vehicle as M_{speed}, and the speed of oncoming vehicles as O_{speed}. Then the relative speed R_{speed} is calculated using formula (1).

$$R_{speed} = M_{speed} + O_{speed} \quad (1)$$

As this study considers grasping the speed on general roads, the ranges of M_{speed} and O_{speed} are as follows:

$$0 \leqq M_{speed} \leqq 60 \quad (2)$$

$$0 \leqq O_{speed} \leqq 60 \quad (3)$$

Then the range of R_{speed} is as follows:

$$0 \leqq R_{speed} \leqq 120 \quad (4)$$

The relative speed is acquired from the Optical Flow Length DB by querying the database with the LOF newly calculated from images. The system estimates the speed of oncoming vehicles by subtracting the speed of the moving vehicle from the relative speed.

However, the relative speed corresponding to the LOF specified in a query might not be stored in the database because the relative speed is continuous, not discrete. The system provides an interpolation function by using LOFs stored in the database. Then the relative speed corresponding to any LOF can be calculated by using the function. The interpolated value is returned as a relative speed when LOF that is not stored in the database is given to the function as an argument.

As we described previously, LOFs fluctuate according to the distance between an in-vehicle camera and an oncoming vehicle, the relative speed, the speed of the moving vehicle, and the speed of an oncoming vehicle. The distance between the in-vehicle camera and oncoming vehicles is shorter than that between the camera and objects behind oncoming vehicles. Therefore, the LOF generated from oncoming vehicles is longer than that generated from background objects by the vehicle's moving. Consequently, the speed of the moving vehicle is higher than interpolated relative speed. In this way the system can distinguish oncoming vehicles from background objects and detect oncoming vehicles.

Figure 6: ROI of generating optical flows

4.3.2 Detecting Traffic Congestion

The causes of traffic congestion are detected considering estimated the speed of oncoming vehicles (SOV). The LOF is calculated, and SOV is estimated for each image when the system detects oncoming vehicles. According to the VICS's classification of traffic congestion, the speed of vehicles in a congested public highway is 10 [km/h]. Therefore, the system judges the location of congestion to be a location in which SOV is continually estimated to be less than 10 [km/h]. At the same time, the image of where congestion begins is a few images before that of the location where an oncoming vehicle is detected at the beginning. In addition, the system considers a location where the SOV is continually estimated to be less than 0 [km/h] or more than 10 [km/h] as the end of the congestion. At this time, IDs of images taken at the beginning and ending point of traffic congestion are saved. The system queries the Image DB with the saved IDs, and acquires the location information of the beginning and ending point of the congestion and an image representing the cause of the congestion.

5 EXPERIMENT AND DISCUSSION

5.1 Experiment Environment

In order to evaluate the effectiveness of our proposed method, we conducted two experiments. First, we conducted an experiment for determining the statistical value of optical flows stored in the Optical Flow Length DB. In addition, we conducted an experiment for evaluating the accuracy of SOV estimation for detecting the traffic congestion in an oncoming lane by using videos recorded in the actual environment.

5.2 Experiment for Evaluation

To generate optical flows and to estimate SOV, we implemented a program with OpenCV libraries. The program reads images, generates optical flows between two successive images, and draws the optical flows onto output images. To calculate optical flows, we used "cvGoodFeaturesToTrack" method which finds the most prominent corners in the image in OpenCV libraries. To calculate optical flows, we used "cvCalcOpticalFowPryLK"

method which is an iterative Lucas-Kanade method using an image pyramid. A vehicle is equipped with an iPhone 3GS placed on the dashboard with a cellular phone cradle, to record video as the vehicle moves. We call this vehicle a 'recording vehicle'. The resolution of videos recorded by the iPhone 3GS is 640×480 pixels, and the frame rate of the videos is 30 [fps]. The rectangular ROI sized 430×210 pixels is placed at the bottom right of images so that oncoming vehicles fit into the ROI, and optical flows are generated within the ROI. To grasp the speed of the recording vehicle, we used the GPS sensor of an iPhone 5 and implemented an iOS application that calculates the speed of the recording vehicle by acquiring location information. To estimate the SOV, we defined three dimensions spline function as an interpolation function by using the dataset of the relative speed and LOF stored in the Optical Flow Length DB. The system gives the spline function with LOF as an argument, and acquires the relative speed. Then the system calculates SOV by subtracting the speed of the recording vehicle from that of the relative speed. In the experiment to determine parameters, four parked oncoming vehicles made congestion on a single lane. In addition, the distance between the vehicles is changed because the distance in actual traffic congestion is non-constant. The driver drove the recording vehicle and past the four parked vehicles five times at different speeds each the inter-vehicular distance, while the iPhone 3GS recorded the oncoming vehicles. Relative speed was equivalent to the speed of the recording vehicle because the oncoming vehicles were parked. We examined parameters such as the time interval between two successive images and the statistical value of LOF for generating optical flows considering the 15 recorded videos. In addition, we defined the interpolation function with parameters derived from the results of the preliminary experiment and the datasets of relative speed and LOF in the database. We considered estimation accuracy with the interpolation function.

5.2.1 Deciding Parameters for Generating LOF

We describe the result of the determination of parameters for generating LOF. Determined parameters are the time interval between two images in the video (*Interval*), and the statistical values of LOF (*Len*) stored in the Optical Flow Length DB. Table 4 shows *Interval* and *Len* considered in this experiment.

The LOF generated in an image is counted, and *Len* is calculated. The values of Interval are 2, 3, 4, and 8. Interval between successive images is 1 when a video is divided into multiple images. For example, if *Interval* is 2, LOF is generated between the nth image and (n+2)th image. It is desirable that *Len* increases in proportion to increasing of the relative speed and the degree of increase of *Len* is large. LOF is acquired when a recording vehicle passes beside the lead oncoming vehicle. Figure 7 shows the environment of the preliminary experiment.

Figure 8, 9, 10, and 11 show the changes of *Len* for each *Interval*. The graph (a), (b) and (c) in each figure show *Dist* = 2, 4 and 6 [m] respectively. *Dist* means the distance between two vehicles in an oncoming lane. Selecting *Len* in the same relative speed that has the same value on different *Dist* is desirable because the distance of two vehicles is not constant in actual environment. In Fig. 8, 9, 10 and 11, blue line shows average *(ave*), red line shows standard deviation (*stddev*), green line shows median (*med*), and purple line shows maximum (*max*). But these figures do not include variance (*var*) because *var* is larger than the other *Len*.

Figure 7: The environment of the preliminary experiment

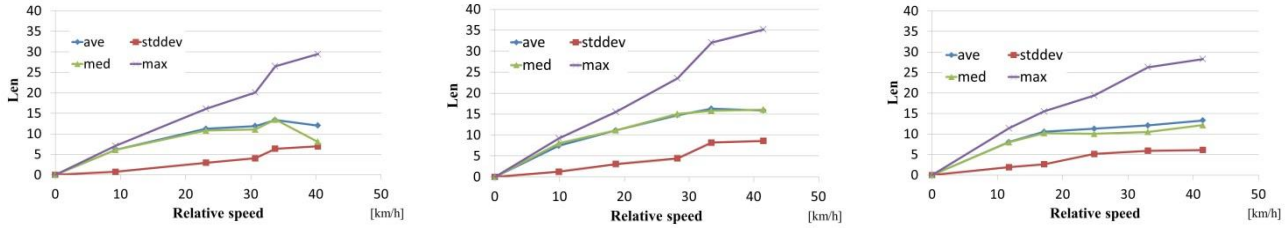
Table 4: *Interval* and *Len* considered in this experiment

The statistical values of LOF (*Len*)	The time interval between two images (*Interval*)
Average (*ave*) Variance (*var*) Standard deviation (*stddev*) Median (*med*) Maximum (*max*)	2,3,4,8

In Fig. 11, *Len* in *Interval* = 8 does not increase monotonically depending on the increase of the relative speed. We suppose that the movement of objects between two images is too large in *Interval* = 8, and the feature points detected in a previous image may disappear in the next image, causing extraordinary LOF to be generated. Consequently, *Len* in *Interval* = 8 is not appropriate to generate optical flows.

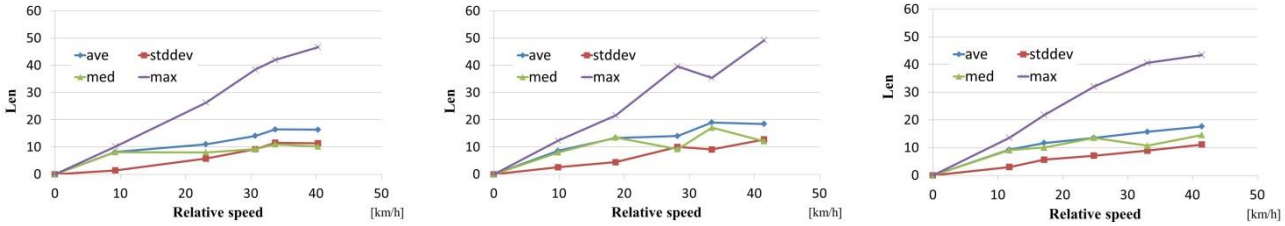
Figure 12 shows the changes of variance (*var*) each *Interval*. In Fig. 12, blue line shows *Interval* = 2, red line shows *Interval* = 3, green line shows *Interval* = 4, and purple line shows *Interval* = 8. *var* fluctuates widely in *Interval* = 3 and 4 shown in Fig. 12. In *Interval* = 2, *var* in the same relative speed are different in each the distance between oncoming vehicles. In addition, standard deviation (*stddev*) is similar to Fig 12. We suppose that *var* and *stddev* are influenced greatly by the false detection of feature points. Therefore, variance and standard deviation are not appropriate to *Len*.

In *Interval* = 2, average (*ave*) and median (*med*) stand still regardless of the increase of the relative speed. In addition, maximum (*max*) increases depending on the increase of the relative speed, and the increased amount of it is small. We suppose that estimating SOV becomes susceptible to the noises of calculating optical flows if increased amount of LOF is small. In *Interval* = 3, *ave*, *med* and *max* increase depending on the increase of the relative speed, and the increased amount of *ave* and *med* are small. In addition, *max* increases with limited influence of the inter-vehicular distance. In *Interval* = 4, *ave* and *max* increase depending on the increase of the relative speed along with Fig. 11, and the increased amount of *ave* is small. *Max* in the same relative speed is different in each inter-vehicular distance, and *med* fluctuates as the relative speed increases.



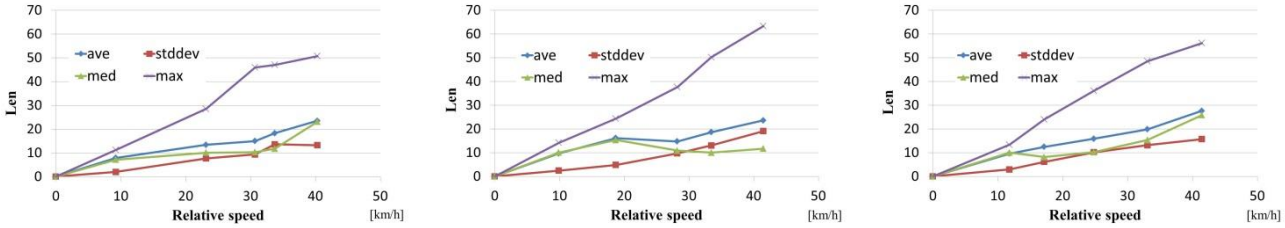
(a) *Dist* = 2 [m] (b) *Dist* = 4 [m] (c) *Dist* = 6 [m]

Figure 8: The changes of *Len* in *Interval* = 2



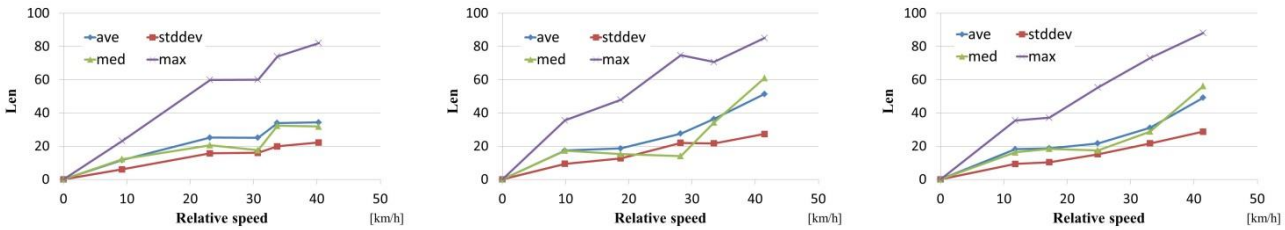
(a) *Dist* = 2 [m] (b) *Dist* = 4 [m] (c) *Dist* = 6 [m]

Figure 9: The changes of *Len* in *Interval* = 3



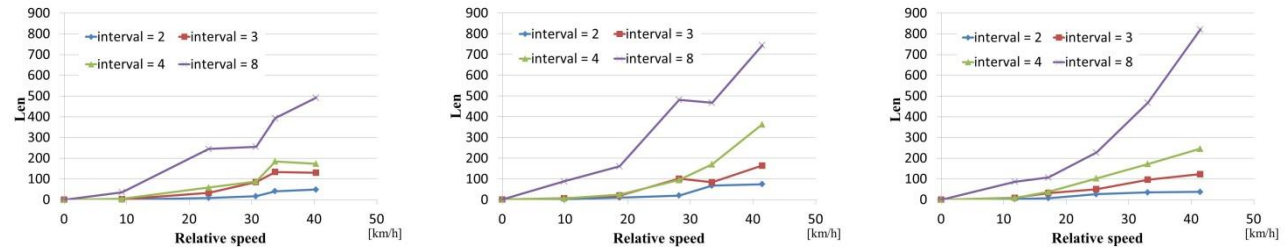
(a) *Dist* = 2 [m] (b) *Dist* = 4 [m] (c) *Dist* = 6 [m]

Figure 10: The changes of *Len* in *Interval* = 4



(a) *Dist* = 2 [m] (b) *Dist* = 4 [m] (c) *Dist* = 6 [m]

Figure 11: The changes of *Len* in *Interval* = 8



(a) *Dist* = 2 [m] (b) *Dist* = 4 [m] (c) *Dist* = 6 [m]

Figure 12: The changes of *Len* (*var*) in each distance

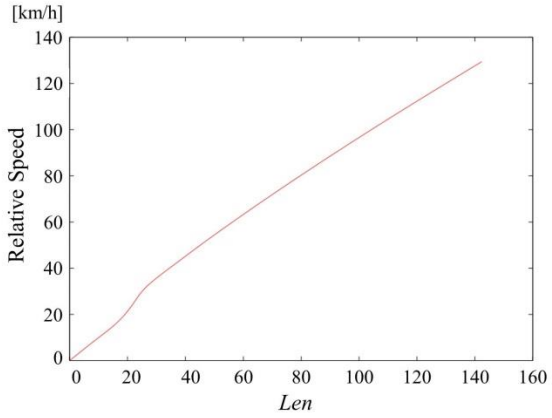


Figure 13: The interpolation function

From these results we can deduce that maximum is appropriate to a statistic of LOF stored in the Optical Flow Length DB (*Len*). In addition, *Interval* = 3 is the candidate parameters for an estimation of SOV experiment. We made the interpolation function by using the dataset of relative speed and LOF acquired through this experiment. However, relative speed does not necessarily increase depending on the increase of *Len*. We suppose that *Len* is influenced by extraordinary LOF results caused by false detection of feature points. They are generated because the vehicle only once drives passing besides the oncoming vehicles at each speed. Therefore, we redefined the function after removing conflicted data and averaging neighbor data with near values.

Figure 13 shows the interpolation function by using maximum in *Interval* = 3 and the relative speed.

5.2.2 Estimation Accuracy

We evaluated the accuracy of SOV estimation by using datasets of *Len* and relative speed determined in the preliminary experiment. For this evaluation, in the same conditions as the preliminary experiment, a recording vehicle moves in the driving lane at 40 [km/h]. At the same time, oncoming vehicles are parked or moving slowly. The recorded video is divided into multiple images. We defined the interpolation function by using the datasets in Optical Flow Length DB. LOF generated from objects in an oncoming lane is given to the function as an argument. Then relative speed (R_{speed}) is calculated by the function according to the formula (1). SOV (O_{speed}) is calculated according to the formula (2). Figure 14 shows the results of estimation.

In Fig. 14, there are points at which SOV gets near to or surpasses 10 [km/h] with time. These speeds are estimated when the recording vehicle passes beside oncoming vehicles, as in Fig. 15(a). At other points, SOV is less than 0 [km/h]. These speeds are estimated until the recording vehicle passes beside the next vehicle shown in Fig. 15(b).

In addition, we found some factors that increase the estimated speed rapidly through this experiment. We suppose that the extraordinary flows generated by false detection of feature points (shown in Fig. 16) cause the rapid increase of the estimated speed. Figure 16(a) shows that optical flows are generated from background objects and Fig. 16(b) shows that the flows are generated from objects on

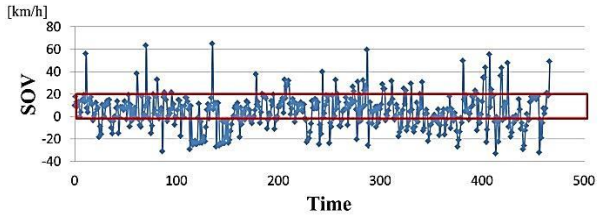


Figure 14: The results of estimation

(a) passing beside an oncoming vehicle
O_{speed} = 12.4
(*Len* = 54.74)

(b) passing beside the next oncoming vehicle
O_{speed} = -13.8
(*Len* = 26.47)

Figure 15: Changes in *Len* and estimated SOV in images

(a) flows generated from background objects

(b) flows generated from other objects

Figure 16: Extraordinary flows generated by false detection of feature points

road except oncoming vehicles. We discuss how the false detection influences the estimated speed in Section 5.2.3.

5.2.3 Discussion of Experiment Results

In this section, we discuss the results of the above experiments. In the experiment to determine parameters, we confirmed that the maximum of LOF is appropriate to generation of optical flows. In addition, we confirmed that the maximum of LOF in *Interval* = 3 increases depending on the increase of the relative speed. However, we confirmed that LOF decreases despite the increase of the relative speed at certain points in each *Interval*. Moreover, we suppose that LOF is calculated accurately by diminishing the false detection of feature points, and the interpolation function that LOF increases depending on increasing of the relative speed can be defined. To diminish the noises of false detection of feature points, we consider the change of the size and position of the ROI and the change of parameters such as the number of detection of feature points. Then, the interpolation function can be defined accurately.

Through evaluating the accuracy of SOV estimation, we confirmed that SOV is estimated near 10 [km/h] when the recording vehicle passes beside oncoming vehicles. In addition, we confirmed that SOV less than 0 [km/h] is continually estimated until the recording vehicle begins passing beside the next oncoming vehicle. We suppose that the reason that LOF remains short is due to the following steps. First, an oncoming vehicle which the recording vehicle passes by slips from an image. Second, feature

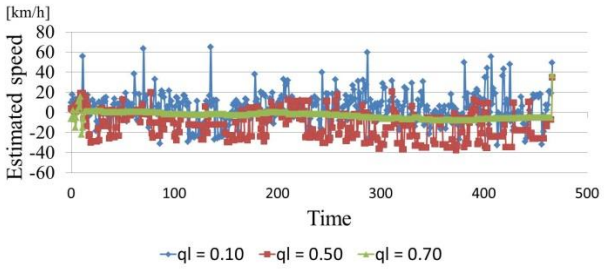


Figure 17: The estimation result in each *ql*

points generated on the oncoming vehicle are insufficient. Finally, feature points are generated anew on the next oncoming vehicle. We noticed certain points at which SOV is less than 0 [km/h] even though the recording vehicle is passing beside an oncoming vehicle. As we have discussed previously in the definition of parameters experiment, the system can estimate the SOV more accurately when processes to define a more accurate interpolation function are applied to the system.

As we explained in Section 5.2, we use "cvGoodFeaturesToTrack" method in the OpenCV library in order to generate optical flows. This method is used for corner detection, and it has the threshold (*ql*) as one of the arguments. *ql* is used to make a judgment on accepting the detected point as corner. When *ql* is larger, sharper feature points are accepted as corner. We examined how the change of *ql* affects the SOV. Figure 17 shows the estimation result in *ql* = 0.10, 0.50 and 0.70.

According to Fig.17, the fluctuation of the estimated speed is larger as *ql* decreases. We suppose that when *ql* is small, the vague feature points are detected as corner and extraordinary flows are generated. In *ql* = 0.70, most of the estimated results are plotted near 0 [km/h], but some of the estimated results are not under 0 [km/h] when oncoming vehicles are not in the image. We suppose that the correct feature points are diminished by increasing *ql*. In the future, we need to consider how to determine the adequate value of *ql*.

The experimental results suggested that the estimated speed changes when a recording vehicle passes by oncoming vehicles. Consequently, detecting the congestion in an oncoming lane will be realized by our method that estimates the speed of oncoming vehicles by using an in-vehicle camera.

6 CONCLUSION

This study aims to detect traffic congestion in an oncoming lane and to present images representing the causes of the congestion using an in-vehicle camera. We proposed a method that estimates the speed of oncoming vehicles using an in-vehicle camera to detect traffic congestion. In addition, we conducted the experiment for estimating the speed and we estimated the speed of oncoming vehicles by using our proposed method. In particular, we suggested that the length of optical flows changes depending on the speed of oncoming vehicles (SOV) and the proposed method based on optical flow is potential to detect the traffic congestion in an oncoming lane. Our method will realize detecting traffic congestion in an oncoming lane based on the results of estimating oncoming vehicles. Consequently, drivers can grasp the causes of the congestion intuitively by images acquired from the results of detecting congestion.

In the future, to improve the accuracy of detection of traffic congestion in an oncoming lane, we will prepare more datasets of LOF and the relative speed. Moreover, we will consider a method for detecting congestion on four-lane roads and divided roads. Our method is only applicable for single lane road. For example, in the case of four-lane roads, LOF generated by oncoming vehicles in each lane is different. In addition, optical flows in divided roads are generated from the median. We will define interpolation functions corresponding to multiple-lane roads and divided roads to improve our method.

REFERENCES

[1] Vehicle Information and Communication System Center, "VICS", http://www.vics.or.jp/index1.html (accessed 4 26, 2015).

[2] H. Kitayama, "New Service and Platform by the Data Utilization from Cars," IPSJ Magazine, Vol.54, No.4, pp.337-343 (2013) (*in Japanese*).

[3] T. Morikawa, "Prospects of Telematics Based on Probe Vehicle Data (<Special Issue> Sophisticated Transportation Systems-Toward Transportation Services to Satisfy Individual Passengers)," Systems, Control and Information Engineers, Transactions of the Institute of Systems, Control and Information Engineers, Vol.54, No.9, pp.366-370 (2010) (*in Japanese*).

[4] J. Bai and L. Zhao, "Research of Traffic State Identification Based on Probe Vehicle," Intelligence Information Processing and Trusted Computing (IPTC), 2010 International Symposium, pp.309-311 (2010).

[5] B. Ramon, G. Javier and S. Joaquin, "Road traffic congestion detection through cooperative Vehicle-to-Vehicle communications," Local Computer Networks (LCN), 2010 IEEE 35th Conference, pp.606-612 (2010).

[6] E. Takimoto, T. Ohyama, R. Miura and S. Obana, "A Proposal and Consideration on a Management Method of Surrounding Vehicle in Vehicle-to-Vehicle Communication Systems for Safe Driving," The Special Interest Group Technical Reports of IPSJ, ITS, Vol.2009, No.24, pp.47-51 (2009) (*in Japanese*).

[7] Y. Xu, Y. Wu, J. Xu and L. Sun, "Multi-hop broadcast for data transmission of traffic congestion detection," Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia (MUM '11), pp. 100-108 (2011).

[8] M. Tamai, K. Yasumoto, T. Fukuhara and A. Iwai, "Efficient Collection and Delivery of Video Data for Traffic Monitoring Utilizing Transition Rate of Congestion Situations," The Special Interest Group

Technical Reports of IPSJ, Vol.2012-MBL-61, No.29, pp.1-8 (2012) (*in Japanese*).

[9] M. Tamai, K. Yasumoto, T. Fukuhara and A. Iwai, "An Image Processing-based Method for Efficient Collection and Sharing of Video Data about Conditions of Vehicular Traffic Congestion," The Special Interest Group Technical Reports of IPSJ, Vol.2012-MBL-65, No.36, pp.1-8 (2012) (*in Japanese*).

[10] H. Kutoku, D. Deguchi, T. Takahashi, Y. Mekada, Ichiro Ide and Hiroshi Murase, "Detection of General Obstacles by Subtraction of Road-Surface with Past In-Vehicle Camera Images," IEICE Technical Report Vol.109, No.470, pp.235-240 (2010) (*in Japanese*).

[11] K. Hamao, Y. Suzuki, M. Honma, K. Hashimoto, Y. Ishikawa, T. takahi, S. Ishiyama and T. Sakurai. "Methods for detection opposite lane traffic jam using a Smartphone," Proceedings of the ITS IEICE, Vol.112, No.72, pp.19-24 (2012) (*in Japanese*).

[12] OpenCV.jp, "Optical flow, http://opencv.jp/sample/optical_flow.html (accessed 4 26, 2015).

[13] G. Bradski and A. Kaehler, Learning OpenCV Computer Vision with the OpenCV Library, California: O'Reilly Media (2008).

(Received November 18, 2014)
(Revised March 2, 2015)

Ryo Shindo received his B.E. and M.E. degrees in information science from Future University Hakodate, Japan in 2013 and 2015. His research interests include probe information system, image processing and ITS. He currently works in Hokkaido CSK Corporation.

Yoh Shiraishi received doctor's degree from Keio University in 2004. He is currently an associate professor at the Department of Media Architecture, School of Systems Information Science, Future University Hakodate Japan. His research interests include database, mobile sensing and ubiquitous computing. He is a member of IPSJ, IEICE, GISA and ACM.

A Simulator for the Execution Efficiency Measurement of Distributed Multi-Database Virtualization

Daichi Kano*, Hiroyuki Sato*, Jun Sawamoto*, and Yuji Wada**

* Graduate School of Software and information Science, Iwate Prefectural University, Japan
** Department of Information Environment, Tokyo Denki University, Japan
sawamoto@iwate-pu.ac.jp

Abstract –In database virtualization technology, the database of a different kind can be used as if it were a kind of database. However decline of execution efficiency is left as one of the research subjects. In improving the execution efficiency, it is necessary to measure the execution performance of the virtualization processes, especially in a distributed environment where multiple databases are connected via a network. In this study, we have designed and implemented the simulator for the execution efficiency measurement. This simulator measures the execution efficiency by calculating the processing time of virtualization processes, database processes and communication processes, and totaling them.

Keywords: Distributed database, Multi-database virtualization, Simulator, Performance evaluation and improvement.

1 INTRODUCTION

Today, it is important to discover and analyze the knowledge and trends which are hidden in large collections of data on ubiquitous network environment using data mining technology, and to use them for decision making of business, etc. However, since those data exists in various types of distributed databases, an appropriate database has to be chosen from a variety of databases and accessed properly. The work of the preparation process of data mining of acquiring appropriate data is needed, and it becomes a burden for the data analysis engineer who performs data mining in the distributed database environment.

To reduce this burden, the multi-database virtualization technology which enables a user to access various types of databases as if accessing a single type of databases has been studied [1-3]. The usefulness has been shown when database virtualization technology is used to perform data mining.

However, some research issues are pointed out. Degradation of the execution efficiency by virtualization processing among the research issues remain by the previous work as one of the main subjects to be solved. Since virtualization processing is performed in addition to normal database processing, it causes execution degradation. Virtualization processing transforms commands and the processing result based on the schema, and when especially the processing result becomes extensively large, virtualization processing becomes a burden. An improvement can be expected by using load sharing technology and parallel processing technology for this issue. While each load decreases by distributing data processing and parallel processing, we anticipate the generation of network delay by low line speed, congestion, etc. Therefore, factors about the network, such as communication time and transmission speed, become important as well as processing of databases.

In improving the execution efficiency, it is necessary to measure the execution performance of the virtualization processing, especially in a distributed environment where multiple databases are connected via a network. But it takes a lot of databases and large-scale network structure, and preparation of actual measurement environment is costly and very difficult. Therefore, the measurement environment using a simulator is considered.

In this study, we have designed and implemented the simulator for the execution efficiency measurement. This simulator measures the execution efficiency by considering the processing time of virtualization processes, database processes and communication comprehensively. And we aim to contribute to quantitative verification and evaluation of the execution efficiency improvement technique of virtualization processing.

The rest of this paper is organized as follows: In section 2, we describe related works. In section 3, we present our proposed solution for database virtualization. In section 4, details of the design of the proposed simulator are described. In section 5, we report the process and some results of acquiring reference parameters for the time of virtualization processing. Finally, the paper is concluded in section 6.

2 RELATED WORKS

The performance estimation of database system is an active research area. They mainly approach this subject by building performance models of database servers and running the models for the simulation [1]-[3].

Garcia [1] presents a simple model based on the queuing network paradigm using fixed distribution for the service times of the queues. The parameters used in the model are adjusted using measurements taken from real servers. This work demonstrates that extreme simple model is capable of predicting the performance of metrics of real database servers with high accuracy and capturing the essential performance aspects of database servers.

Wu, et al [2]-[3] propose a method for predicting query execution time for concurrent and dynamic database workloads. Their approach is based on analytic model rather than machine-learning model. They use optimizer's cost model to estimate the I/O and CPU operations for each individual query, and then use a queuing model to combine

these estimates for concurrent queries to predict their execution times. A buffer pool model is also used account for the cache effect of the buffer pool.

These related works are all targeted for real database servers. On the other hand, our target is virtualized distributed multi-database system. And we have designed and implemented the simulator for the execution efficiency measurement by considering the processing time of virtualization processes, database processes and communication processes. Our main goal is to discover the bottlenecks of the database virtualization processing.

Some earlier reports [4]-[6] have described the study of database virtualization technology.

Mori et al. [4] proposed development of a system to disseminate information actively to all users in a mobile computing environment. They implemented an experimental system using the meta-level active multi-database system as the platform in a mobile computing environment. By mapping the data of the local database group to a meta-database through the basic search and build operations, the system intends to combine data and include different types of local database group.

The data integration technique, Teiid [5], enables virtualization of various types of databases; through such virtual databases, one can access such data sources as relational databases, web databases, and application software such as ERP and CRM, etc. in real time. They can all be integrated for use. In fact, Teiid has a unique query engine. Furthermore, the real-time data integration is accomplished by connecting business application software through the JDBC/SOAP access layer with data sources which are accessed through the connector framework.

In [6], they similarly describes a module known as a wrapper that allows accessing and integrating data from various sources such as RDBs, the Web, and Excel files.

In our previous study [7], we considered the metadata, UML, ER model, and the XML schema as candidates for use to accomplish database virtualization. Thereby, ubiquitous databases can be used as if they were a single database. We then compared the advantages and disadvantages of each to analyse them as follows.

In our previous studies [7]-[10], we examined XML schema advantages and proposed a virtualization method by which such ubiquitous databases as relational databases, object-oriented databases, and XML databases are usable, as if they all behaved as a single database.

3 DATABASE VIRTUALIZATION [7]

Databases of many kinds exist in terms of their associated data model differences and vendor differences. Regarding differences among data models, each has different data representation, and unique associated manipulation. Some typical examples include the table type of relational databases (RDB), XML-representation type of XML databases (XMLDB), and object-oriented databases (OODB). Even the same model database might have different features among vendors. Regarding RDB for example, there might be some differences in SQL and/or data type representation. The typical example is that we have MySQL, PostgreSQL, and SQLServer from different vendors.

These differences according to the model and vendor bring some undesired results. For example, we might end up spending more time and labor during application system development because of the different data models that must be confronted. For example, we might need to acquire the right API to handle data of every different type of database. Virtualization of such different types of modelled databases to unify the procedures for all of them would probably impart less of workload and cost, and facilitate their management in a more flexible manner. Consequently, virtualization of databases, if it could be done, would facilitate application system design and database management as well.

To have a virtualization feature, we will consider the inclusion of features to manage distributed databases of similar types, the distributed databases of different types, and provide location transparency for users, such that they notice no differences of database structure or location and become able to use databases of all kinds in a flexible fashion. Fig. 1 portrays an example view of the database virtualization technique.

For virtualization of ubiquitous databases in our study, we will describe the schema information of the real databases, of which more than one always happens to exist, by creating and using one common XML schema. We also provide functionality of data search and update with the XML-based common data manipulation API.

3.1 XML Conversion Program

We will use an XML schema that provides a flexible representation capability and a high transparency capability.

To do so, we will produce such a virtualization concept in which the user would feel as if he or she were locally manipulating the remote site RDB from a local RDB process environment. That can be accomplished by converting the schema information and data information of the local RDB into the XML schema, and then storing that information into the RDB that the user would like to operate.

We developed an XML conversion program, XML Export/Import, as depicted in Fig. 2. We then used such different vendor RDBs as MySQL, PostgreSQL, and SQLServer2005 because they are available in the RDB virtualization system creation environment. We have to rebuild the XML tree with our XML conversion program when the distributed database is redefined.

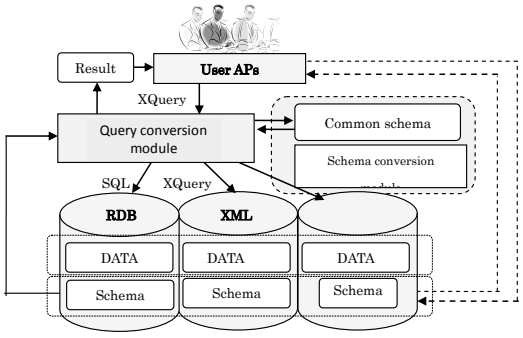


Figure 1: An example view of database virtualization.

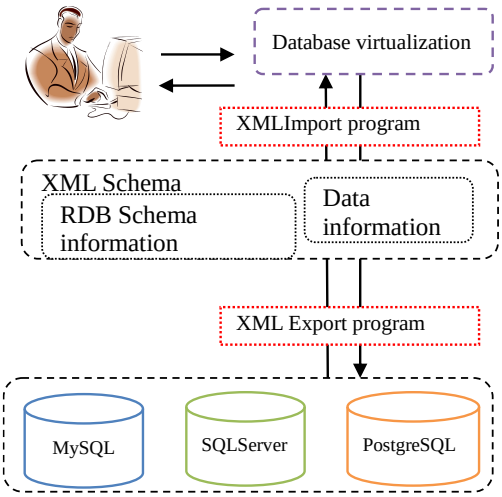


Figure 2: Virtualization technique for RDB databases.

3.2 RDB Schema Conversion into XML

The following describes how the RDB schema is converted into XML. Fig. 3 presents results of reading the schema information from the RDB and converting it into XML. The RDB schema information that is converted into an XML format includes "table names", "field names" (associated data types and default values), and "constraints" (primary key constraint, unique constraint, check constraint, NOT NULL constraint, and foreign key constraint) capability.

Regarding the XML tree structure, we described the table information in the table structure node with its elements of Field="column name", Type="data type", Null="TRUE or FALSE" (NOT NULL constraint). We described the schema information in the schema node with its elements of TYPE= "constraint name", Table= "table name", Column= "column name", ReTable= "referenced table name", ReColumn= "referenced column name", and Check= "rule".

3.3 RDB Data Conversion into XML

The manner in which the RDB data are converted into XML is described next. Fig. 4 portrays results of reading the data information from the RDB and conversion into XML. Because of the XML tree structure, we had dbname="database name", tblname="table name", and the actual data columns succeed.

3.4 Virtualization of Databases

We discuss the virtualization of modelled DBs of different types. For virtualization of different types of modelled DB, we describe the schema information of each model using a single common schema. The common schema we will use is an XML Schema. Around it, we will perform virtualization. Fig. 1 shows a virtualization method for different database types. To accomplish schema conversion from a different modelled database, we first get the schema information from an RDB to work on. Then we convert it into the correct XML schema for that RDB. We currently have to re-build

```
<?xml version="1.0" encoding"UTF-8" standalone="yes"

 <root>

<rdb Name="mysql">

   <database Name="questionnaire" >
      <table_structure Name="member">
      <field Field="samplenum"
              Type="integer" Null="FALSE"  Default="  />
   <field Field="answerday" Type="text"
     Null="FALSE"  Default="  />

   ….

 </table_structure>

 <schema>
  <constraint Type="PRIMARY KEY"
      Table="member"  Column="samplenum"  />

….

      <constraint Type="FOREIN KEY" Table="questionnaire"
      Column="samplenum" Retable="member"
      ReColumn="samplenum" />

….

 </schema>
</database>
</rdb>
</root>
```

Figure 3: Example of RDB schema information conversion into XML.

Code	Name	Latitude	Longitude
47401	Wakkanai	45.25	141.41
47404	Haboro	44.22	141.42
…	…	…	…

RDB

```
<?xml version="1.0" encoding="utf-8" ?>
<dataset dbname="chihou">
    <data tblname="AreaInfo ">
      <Code>47401</Code>
      <Name>Wakkanai</Name>
      <Latitude>45.25</Latitude>
      <Longitude>141.41</Longitude>
   </data>
     <data tblname=" AreaInfo ">
      <Code>47404</Code>
      <Name>Haboro</Name>
      <Latitude>44.22</Latitude>
      <Longitude>141.42</Longitude>
   </data>
   <…>
```

Figure 4: Example of actual RDB data conversion into XML.

the XML tree with our schema conversion module when the distributed database is redefined.

Table 1 presents schema conversion correspondences between the two. Because any XML DB is already described in the XML format, we extract the schema information without conversion. On the other hand, when the data are manipulated, our query conversion module automatically transfers the access results to the application program.

3.5 Techniques of Execution Efficiency Improvement

Methods of the execution efficiency improvement of virtualization processing (improvement in the speed) are as follows.

- The place of virtualization processing

In order to accelerate, the virtual database environment which uses load sharing technology and parallel processing technology is shown in Fig. 5, and we use both user side virtual DBMS and data side virtual DBMS.

Since the database is distributing through a network and communication time influences the whole processing time greatly, it becomes important to reduce the amount of data transfer for the improvement of the processing speed. Under the virtualization processing the data volume changes. Even if the same data is processed, data volume differs by the schema expression, RDB schema or XMLDB schema. Therefore, the place where the virtualization processing is performed could be changed, so that the amount of data transferred is reduced, and communication time is reduced.

- Database selection

When the same table and data are stored in different databases, it could be considered to make the load of each database uniform by acquiring data from a database with little load. In database virtualization technology, since virtual processing is added in addition to processing of the usual database, balancing of the database load becomes important.

4 DESIGN OF THE SIMULATOR

In this section, we design a simple model based simulator based on the database virtualization technique described in Section 3. Only two types of DBs, e.g. RDB and XMLDB, are considered here.

In improving the execution efficiency, it is necessary to measure the execution performance of the virtualization processing, especially in a distributed environment where multiple databases are connected via a network. But it takes a lot of databases and large-scale network structure, and preparation of actual measurement environment is costly and very difficult. Therefore, the measurement environment using a simulator is considered.

Two of the followings are the basic requirements needed by the simulator.

- Measurement for discovering the causes (bottlenecks) of delay of database virtualization processing can be performed.
- Measurement when the number of databases connected or the volume of each database becomes large on the virtual database environment using load sharing technology and parallel processing technology can be performed.

Table 1: SQL and associated XML

	SQL	XML
Table definition	CREATE TABLE table name…	<xsd: element name="table name"…
Column definition	CREATE TABLE… column name...	<xsd: element name="column name"…
Data type definition	CREATE TABLE… data type..	<xsd: element… type="data type"…
Default values	CREATE TABLE… column name DEFAULT value	<xsd: element… default="value"…
Primary key constraint	PRIMARY KEY	<xsd: key…
Unique constraint	UNIQUE	<xsd:unique …
Foreign key constraint	FOREIGN KEY	<xsd: keyref … refer =…
NOT NULL	NOT NULL	<xsd:… nillable="false"...
Method	CREATE METHOD	
Inheritance	CREATE TABLE… UNDER upper level table name	<xsd: complexType …

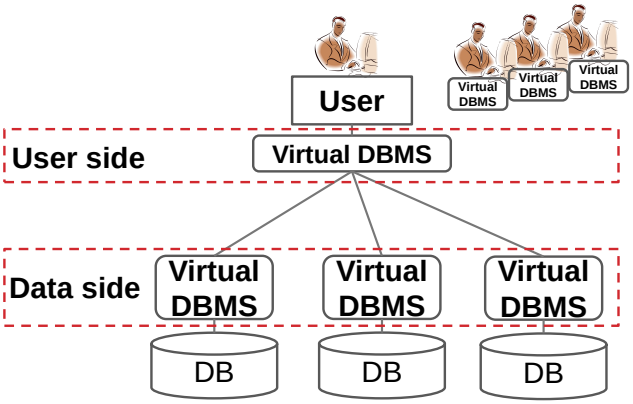


Figure 5: Virtual database environment which uses load sharing technology and parallel processing technology.

4.1 Outline of the Simulator

The main purpose of the simulator is the bottleneck discovery of database virtualization processing. For the purpose of this bottleneck discovery, actual processing, such as virtualization processing, database processing and communication processing, are not needed and actual processing is not performed in the simulator. The execution efficiency is computed simulating and integrating each processing time. Random elements such as network

congestion, user's command input timing are simulated and computed repeatedly to obtain average and variance.

Prerequisites for database access for the simulation are specified as follows. The data mining and distributed database environments are considered in the simulator and it assumes a limited range of database operations here. For example, database updating and join operations are excluded in the simulator.

By realizing each component such as database processing of the simulator as a process and performing inter-process communication with TCP protocol, the simulator can be implemented on a single PC or on two or more PCs. Followings are prepared as an item which can be changed by setup.

- Number of users
- Number, scale, and kind of databases
- Network line speed

4.2 Measurement Items

The following measurement is performed for the overhead identification of virtualization processing. About the reference parameters for the simulation, some preliminary simple virtual processings are performed beforehand and they are determined from the result at the time of implementation.

- Time of virtualization processing

This mainly considers time of conversion such as query conversion from XQuery to SQL and result conversion from RDB result into XML format. The measuring method computes and converts the processing time according to the length of a query, the data volume of the result, etc. based on the reference parameters.

- The change in the data volume after virtualization processing

The data volume fluctuated by virtualization processing of query result is measured.

- Processing time of a database

The processing time of a database is computed from a query. For example, in 'Selection', processing time changes by the existence of indexes. Processing time is changed also by the timing of the database usage and the number of users. If there are some database processing performed during system usage of a user, the wait time of the database processing will be added to the processing time for the user.

- The amount of data transfer

The data transfer rate is adjusted by changing the network utilization factor according to the number of users, users' usage timing, etc. of databases. The system determines the amount of data volume by what kind of query is issued to which database by each user, then decides the amount of data transfer by which network is used for the data transfer.

- Communication time

Communication time is computed using the following formulas.

$$Communication\ time = \frac{Amount\ of\ data\ transfer \times (1 + Rate\ of\ control\ data)}{Line\ speed \times Network\ utilization\ factor}$$

Since the network of a database is classified to class 3 in Network Quality of Service (QoS) of Y.1541 of ITU, delay by congestion is generated in the probability of 10^{-3} based on the class 3 of QoS. Time to be delayed in this case, being unspecified in the class 3 of QoS and not restricted, we make it the interval of the retransmission-of-message packet. The process on the data reception side performs the measurement of communication time.

4.3 Size of Packet

Packet size is needed for the determination of the rate of control data or the number of times of communication. The maximum size (MSS: Maximum Segment Size) of the packet changes with MTU (Maximum Transmission Unit) of the data link assuming that the database uses TCP.

The main current data links are Ethernet and PPPoE, and assuming the protocol uses TCP, MTU of Ethernet is used.

The maximum data volume per packet is set to 1460 bytes, and the number of times of communication is (Amount of data transfer /1460) and the rate of control data is (1-1460 / 1518).

4.4 System Configuration

Each component is realized by a process so that the each component, such as virtual DBMS, can be executed concurrently. Each component performs inter-process communication with TCP protocol, and the simulator is run on a single PC or two or more PCs. Development language is C and execution environment is Linux.

In order to decide to implement virtualization on user side or data side depending on the measurement result, virtualization process could be performed on both sides. Although designed supposing virtualization of RDB and XMLDB at this time, when adding virtualization of other DB kinds, it is made to be easy to extend. By saving the last setting environment in a file, and calling it easily, the time and effort for the setup for every simulator use is reduced.

The system configuration of the simulator based on Fig. 5 is shown in Fig. 6. And the component processes of the simulator are classified into following three.

- Interface process for the simulator user

Processing of a simulator user's interface and management of the whole simulator are performed. The setup of the simulator and directions of a simulation start are performed.

- User's process

Processing corresponding to each user using a database is performed. Execution of XQuery, reference of an XML schema, etc. are performed and processing time is sent to the interface process for the simulator user. In a communication module, calculation and conversion of communication time are performed from the data volume of the received result. In a virtualization process module, calculation and conversion of time of virtualization processing from the data volume of a result are performed.

- Handling process of each database

Processing of data side virtual DBMS and database accesses are performed. The processing time for processing of a database and virtualization processing according to a setup of the number of data etc. is computed and converted.

In a communication module, calculation and conversion are performed for the communication time of query reception based on the received query. By DB module, calculation and conversion of the processing time concerning query execution are performed and data size or the number of data of the result data are determined. In the virtualization process module, calculation and conversion of time for virtualization of data from the number of result data, etc. are performed.

In a communication module, since a transmitting side process does not need to consider the existence of delay, such as a collision, about measurement of a communication time, the communication module of the receiving side process measures communication time. Specifically, measurement of communication time in case a command is sent to data side virtual DBMS from user side virtual DBMS is performed by the database side communication module and in case a result is sent to user side virtual DBMS from data side virtual DBMS, measurement is performed by the user side module.

5 REFERENCE PARAMETERS FOR THE TIME OF VIRTUALIZATION PROCESSING

In this section, we determine necessary reference parameters for the simulator model in Section 4. The parameters are determined using measurements taken from real virtualization processing and database access.

Simple and preliminary virtualization processing was performed and the reference parameters of the processing time of virtualization processing and the fluctuation of the data volume after virtualization processing were determined. Although implementation was carried out in Java by the previous work [7], since Java operates on a virtual machine and delay by insufficient memory occurs, we re-implemented the system in C.

At this stage, since database virtualization of only RDB and XMLDB is assumed, only the reference parameters of these virtual processings are obtained. Moreover, execution using an actual database is not performed about processing of a database, but the function which returns dummy data is prepared. The execution environment of preliminary virtualization processing is as shown in Table 2.

The reference parameters obtained in this section are the references only for the environment shown in Table 2. The reference parameters should be reconsidered and modified under other environments.

About the composition of a database, RDB 'Chihou' assumes the database with the table and column shown in Table 3, and assumes the XMLDB database 'Tenkou' which is shown in Fig. 7.

The XQuery used for the execution is as follows.

```
for $A in fn:doc('Tenkou')//Item let $B :=
fn:doc('Chihou')//areainfo[@Code=$A/Station/Code]
let $C := fn:doc('Chihou')//observ[@Code=$A/Station/
Code] return <result>{$B/@Code, $B/Area, $B/Kana,
$C/Observ, $A//Precipitation, $A//Precipitations}
</result>
```

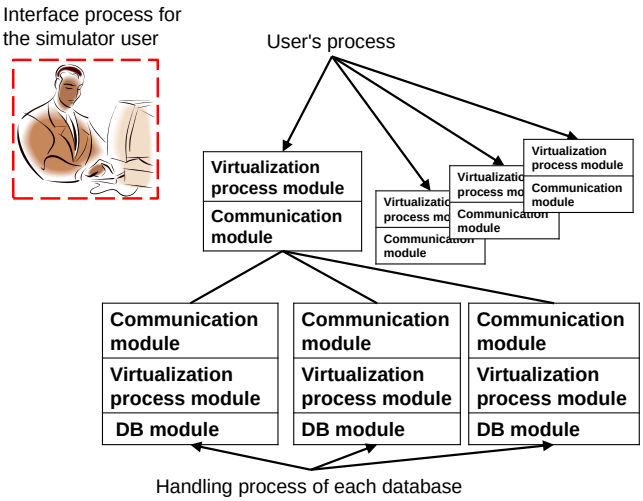


Figure 6: System configuration.

Table 2: Preliminary virtualization process execution environment

OS	Windows 8.1 pro 64bit
CPU	Core i5-3317U1.70GHz 2threads
Memory	4GB

Table 3: Structure of RDB 'Chihou'.

Table name	Column name
Areainfo	Code, Area, Kana
Observ	Code, Observ

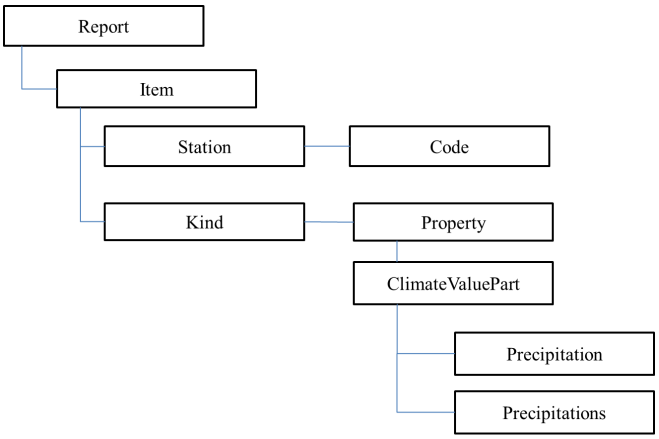


Figure 7: Structure of XMLDB 'Tenkou'.

Table 4: The virtualization processing time of the execution result of the query of RDB (microseconds).

	Number of columns		
Number of result data	1	3	5
500,000	657,763	1,660,663	2,658,075
1,000,000	1,280,225	3,270,225	5,205,550
2,000,000	2,501,350	6,092,450	10,225,800

Table 5: The virtualization processing time of the execution result of the query of XMLDB (microseconds).

	Number of items		
Number of result data	1	3	5
500,000	855,000	2,285,700	3,721,778
1,000,000	1,714,250	4,554,200	7,202,556
2,000,000	3,398,000	8,741,800	14,397,000

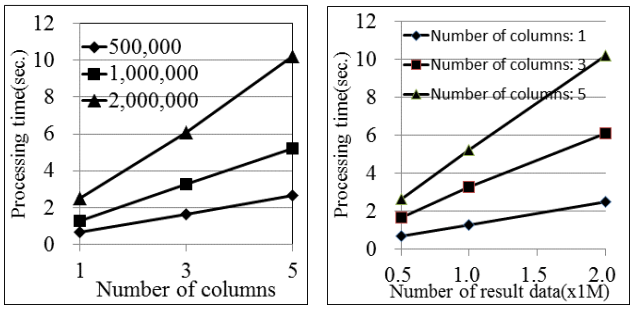


Figure 8: Graph of the virtualization processing time of the execution result of the query of RDB

This XQuery is a query which acquires data from RDB named 'Chihou' and XMLDB named 'Tenkou'. It is the query of returning the result which acquired from 'Chihou' of the 'let' phrase based on the result of 'Tenkou' acquired with the 'for' phrase, in the form described after 'return' phrase. From the simple execution result of virtualization processing, virtualization processing of a query execution result has measured time.

To determine the reference parameters, queries for above mentioned processing which return 500,000, 1,000,000 or 2,000,000 result data, are created and executed multiple times. From the execution results, reference parameters are determined as shown in Table 4, 5.

For the virtualization processing time of the execution result of the query of RDB, it is proportional to the number of result data and the number of columns, as shown in Fig. 8. Moreover it can be expressed by a linear equation of 0.953 microseconds of inclination and 0.208 microseconds of intercept of the number of columns. Value by these parameters and actual measurement are shown in Fig. 9.

For the virtualization processing time of the execution result of the query of XMLDB, it is proportional to the number of result data and the number of items, like RDB. Therefore, it can be expressed by a linear equation of 1.393 microseconds of inclination and 0.317 microseconds of intercept of the number of item.

The determined reference parameter of each processing time is shown in Table 6. As mentioned before, these reference parameters are the references only for the environment shown in Table 2. But, the main purpose of our simulator is the bottleneck discovery of database virtualization processing. So, we do not need to know absolute virtualization processing time. We need to know the relative ratio between the virtualization processing time and the database processing time. Although the database processing time is not shown yet in this paper, it should be measured in the same environment as this time, and we could use it.

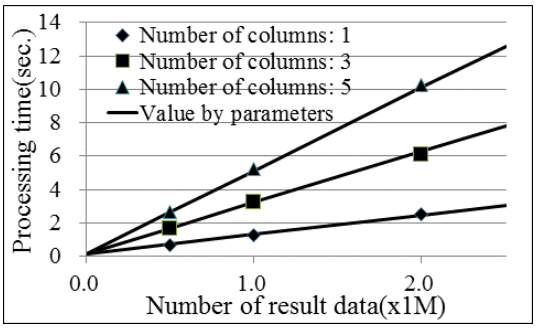


Figure 9: Value by reference parameters and actual measurement of the query of RDB

Table 6: Reference parameter of processing time (microseconds).

Processing	Processing time
Virtualization processing time of the execution result of the query of RDB	(0.953 x Number of columns +0.208) x Number of result data
Virtualization processing time of the execution result of the query of XMLDB	(1.393 x Number of items +0.317) x Number of result data

6 CONCLUSION

In this research, the design and implementation of the simulator which measure the execution efficiency of the database virtualization processing in the distributed environment where multiple heterogeneous databases were connected with the network have been performed.

However, verification and evaluation of this simulator itself is left yet. Therefore, it is necessary to advance to the next stage of performing verification and evaluation of the simulator, and perform quantitative measurement of database virtualization processing. From the result, we discover the bottleneck of database virtualization processing, and plan to accelerate the bottleneck parts in the future.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Number 24500122.

REFERENCES

[1] D. F. Garcia, "Performance Modeling and Simulation of Database Servers." The Online Journal on Electronics and Electrical Engineering Vol.2, No.1, pp.183-188 (2010).

[2] W. Wu, et al., "Predicting query execution time: Are optimizer cost models really unusable?." IEEE 29th International Conference on Data Engineering (ICDE), pp.1081-1092 (2013).

[3] W. Wu, et al., "Towards predicting query execution time for concurrent and dynamic database workloads." Proceedings of the VLDB Endowment, Vol.6, No.10, pp.925-936 (2013).
[4] K. Mori, S. Kurabayashi, N. Ishibashi, and Y. Kiyoki, "An Active Information Delivery Method with Dynamic Computation of Users' Information in Mobile Computing Environments." DEWS2004 1-A-04, (2004). (in Japanese)
[5] Teiid: http://www.jboss.org/teiid, Red Hat
[6] DB2: Information Integrator V8.1, http://www.jpgrid.org/documents/pdf/WORK4/sugawara_ws4.pdf
[7] Y. Wada, Y. Watanabe, K. Syoubu, H. Miida, J. Sawamoto, "Virtual Database Technology for Distributed Database in Ubiquitous Computing Environment," American Journal of Database Theory and Application, Vol. 1, No.2, pp.13-25 (2012).
[8] Y. Wada, Y. Watanabe, K. Syoubu, J. Sawamoto, and T. Katoh, "Virtualization Technology for Ubiquitous Databases," Proc. 4th Workshop on Engineering Complex Distributed Systems (ECDS), pp.555-560 (2010).
[9] Y. Wada, Y. Watanabe, K. Syoubu, J. Sawamoto, and T. Katoh, "Virtual Database Technology for Distributed Database," Proc. IEEE 24th International Conference on Advanced Information Networking and Applications Work-shops (FINA2010), pp.214-219 (2010).
[10] Y. Wada, Y. Watanabe, K. Syoubu, H. Miida, J. Sawamoto and T. Katoh, "Technology for Multi-database Virtualization in a Ubiquitous Computing Environment," International Workshop on Informatics (IWIN2010), pp. 89-96 (2010).

(Received October 23, 2014)
(Revised February 9, 2015)

Daichi Kano received M.S. degree in 2015 from Iwate Prefectural University, Japan. His research interests include distributed parallel processing and simulation. He is currently working for Tokyo Computer Service Co., LTD.

Hiroyuki Sato is currently a Professor of Faculty of Software and Information Science, Iwate Prefectural University, Japan. He received the B.E. in information engineering from Tsukuba University in 1982. He joined Mitsubishi Electric Corporation in 1982. He received his PhD degree from Tsukuba University in 2003. His research interests include parallel processing, and high performance computing. He is a member of IPSJ, IEICE and IEEE-CS.

Jun Sawamoto is currently a Professor of Faculty of Software and Information Science, Iwate Prefectural University, Japan. He received the B.E. and M.E. in mechanical engineering from Kyoto University in 1973 and 1975. He joined Mitsubishi Electric Corporation in 1975. He received his PhD degree from Tokyo Denki University in 2004. His research interests include ubiquitous computing, human-interface system, multi-agent systems, and cooperative problem solving. He is a member of IPSJ, IEEE-CS, ACM.

Yuji Wada received the B.E. and the M.E. in electrical engineering from Waseda University in 1974 and 1976, respectively. He joined Mitsubishi Electric Corporation in 1976. He received the PhD degree in computer science from Shizuoka University of Japan in 1997. He is currently a Professor in the Department of Information Environment, Tokyo Denki University. His research interests include database systems, data mining, and recommendation. He is a member of the IPSJ, the IEICE, the JSAI, the JSSST and the DBSJ.

Formal Verification Technique for Consistency Checking between equals and hashCode Methods in Java

Kozo Okano[††], Hiroaki Shimba[†], Takafumi Ohta[†], Hiroki Onoue[†], and Shinji Kusumoto[†]

[††]Department of Computer Science and Engineering, Shinshu University
[†]Graduate School of Information Science and Technology, Osaka University
{okano, h-shimba, t-ohta, h-onoue, kusumoto}@ist.osaka-u.ac.jp

Abstract - Java objects used with the standard collection should override both of its equals and hashCode methods. Both methods need to satisfy the consistency rules or unexpected behaviors may cause faults that are hard to detect. A previous study checked whether an equals method satisfies part of the consistency rule. To avoid unexpected behaviors, however, it is necessary to check that both the equals and the hashCode methods satisfy the rules. This research proposes a method which checks the consistency between equals and hashCode methods in Java. We model Java source code and check whether both methods satisfy the rules using an SMT solver called Z3. We applied our proposed method to some practical projects. As results, we detected some Java source code that violates the rules.

Keywords: Java, equals method, hashCode method, Formal Verification, Satisfiability Modulo Theories (SMT)

1 INTRODUCTION

In Java, an equals method should be rightly overridden in a class, if its objects are compared. To guarantee the appropriate behavior of the collection framework, when a class overrides its equals method, its hashCode method should also be overridden [1]. Therefore, a document of Oracle API defines some rules for the methods in an Object class [2]. For example, an equals method is necessary to satisfy reflexive, symmetric, and transitive properties. A method violating the rules may cause faults. It is well known that such faults are hard to detect [1][3][4]. Rupakheti et al. [5]-[7] presented a checker called EQ, which is designed to automatically detect an equals method violating the rules. EQ models an equals method and performs model checking to check whether the equals method satisfies part of the rules. Since EQ checks only equals methods, it cannot detect a class that may cause a fault when its object interacts with the collection framework. Also, EQ uses a model description language called Alloy, which cannot model bit operations. Hence, EQ cannot model equals methods using bit operations. To avoid the unexpected behavior, we propose a new method which checks the inconsistency between equals and hashCode methods. We use a Satisfiability Modulo Theories (SMT) solver called Z3 [8] to manipulate arithmetic operations and bit operations which are often used in hashCode methods. Since the implementation patterns of equals and hashCode methods are different, we propose new implementation patterns of hashCode methods. Also, we propose a method which converts Java code to an expression in a model description language called SMT-LIB [9]. We applied our proposed method to some practical projects. As results, we detected some Java source code violating the rules. The rest of this paper is organized as follows. Section 2, Section 3, Section 4, Section 5, Section 6, and Section 7 present the consistency rules for equals and hashCode methods, a details of Z3, a motivating example, how to convert Java code to SMT-LIB, an evaluation of our proposed method and discussion, and the conclusion of this paper, respectively.

2 CONSISTENT RULES

This section presents the rules that equals and hashCode methods must satisfy.

2.1 Java Object Class

The Java Object class is defined as the "root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class." by an Oracle API document [2].

2.2 Consistent Rules for equals Methods

An equals method for Object class determines whether some other object supplied through its argument equals this object. An equals method must satisfy the following four rules except for a null object [2].

- reflexive: for any non-null reference value x, x.`equals`(x) should return true.
- symmetric: for any non-null reference values x and y, x.`equals`(y) should return true if and only if y.`equals`(x) returns true.
- transitive: for any non-null reference values x, y, and z, if x.`equals`(y) returns true and y.`equals`(z) returns true, then x.`equals`(z) should return true.
- For any non-null reference value x, x.`equals(null)` should return false.

The equals method for Object class is defined as follows [2]. "The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true). Note that it is generally necessary to override the hashCode method whenever this method

```
public class Sample{
 private int val;
 private String str;
 public boolean equals(Object obj){
  if (obj == null)
   return false;
  if (this == obj)
   return true;
  if (!(obj instanceof Sample))
   return false;
  Sample that = (Sample) obj;
  if (this.str == null){
    return that.str == null;
  }
   return this.val == that.val && this.str.equals(that.str)
 }
 public int hashCode(){
  return val + (this.str == null ? 0 : this.str.hashCode());
 }
}
```

Figure 1: Example of correct implementation of equals and hashCode methods

is overridden to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes."

2.3 Consistent Rules for hashCode Methods

The hashCode method returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by HashMap. The hashCode method must satisfy the following two rules [2]. In this definition, information implies the returned value from the method invoked by its equals method or a field value used in the equals method. Thus, if some inconsistency exists between equals and hashCode methods, a rule violation occurs.

- Whenever it is invoked on the same object more than once during the execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer does not need to remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the hashCode method on each of the two objects must produce the same integer result.

The hashCode method for an Object class returns a different integer value for each different instance. Figure 1 shows an example of a correct implementation of equals and hashCode. The sample class has val and str as the integer and String type field values, respectively. The equals method for the sample class determines whether an argument is the instance of the sample class after it determines whether an object passed as the argument is identical to itself. Next, if the field value str is null, the equals method checks whether the str in passed object is also null. Finally, it determines whether the value of val and the string of str are identical. The hashCode method for the sample class concatenates the value of val and the hash value of str. The sample class satisfies the consistent rules for both the equals and the hashCode methods.

3 RELATED WORK

Research on the implementation and the design of a method in Object class proposed a method that automatically generates the equals and hashCode methods. Rayside et al. proposed a method which automatically generates the equals and hashCode methods which match the user demands by using an annotation of classes and methods [10]. This study performs a dynamic analysis of source code. Grech et al. solved the problem of the Rayside research, which consumes a long time to verify cyclic objects by analyzing source codes statically [11]. Also, Jensen et al. proposed an annotation which guides the user when the user copies objects by the clone method [12]. Recently, research using model checking by the Boolean Satisfiability Problem (SAT) solver and the SMT solver have gained attention. Anastasakis et al. proposed a conversion method that converts class diagrams of the Unified Modeling Language (UML) with the Object Constraint Language (OCL) to Alloy [13]. This research helps the developer who would like to perform verification about Alloy without knowledge of Alloy. Liu et al. suggested scalable bounded model checking by representing object-oriented languages as a bit vector of the SMT solver [14]. This research supports high-speed verification. Balasubramaniam proposed the constraint solver MINION that has high scalability and provides many functions [15]. Also, they proposed a method that automatically generates a constraint solver optimized to each domain [16]. This research helps the generation of the domain-specific constraint solver. Burdy et al. proposed a method that statically verifies Java source code [17]. This method specifies the code location that may cause exceptions, such as a NullPointerException. Also, it can verify Java source code annotated with JML. It is able to check whether each method satisfies its constraints based on JML.

3.1 EQ

EQ [7] checks whether the equals method in Java satisfies the consistency rules. EQ receives a type hierarchy and outputs whether the equals method satisfies the consistency rules. Hereafter, a type hierarchy is a structure of classes and interfaces represented as a directed acyclic graph (DAG). Except Object class, the classes and interfaces which have an inheritance relation belong to the same type of hierarchy. EQ consists of the following four steps. 1) Perform path analysis for the equals method. 2) Analyze the pattern of the equals method. 3) Convert Java code to a model described as Alloy. 4) Verify the model by an Alloy analyzer. EQ has two problems. One problem is that EQ does not check whether a hashCode method satisfies the consistent rules. The other is that, since Alloy cannot model bit operators, Alloy cannot

```
public class COSString extends COSBase{
   public byte[] getBytes(){
      ...
   }
   public boolean equals(Object obj){
   return (obj instanceof COSString)&&
      java.util.Arrays.equals
      (((COSString)obj).getBytes(),getBytes())
   }
   public int hashCode(){
      return getBytes().hashCode();
   }
}
```

Figure 2: hashCode methods violating consistency rules in PDFBox of Apache

model usual hashCode method using bit operators. In this study, to solve those two problems, we use Z3 not Alloy.

3.2 Z3

The SMT problem is a decision problem for logical formulas expressed in first-order logic. An SMT solver solves SMT problems automatically. The SMT solver determines if a given logic formula, which is a combination of theories expressed in first-order logic, is satisfiable. If the theories are satisfied, the SMT solver outputs assignments for variables that make the given theory satisfied. SAT problems are described as theories that consist of only propositional variables. On the other hand, SMT problems are described as theories that consist of many propositional types, such as Int, which are similar to the types in programming language. Also, SMT problems can define and use functions. In this study, we determine whether both the equals and the hashCode methods satisfy the consistency rules by using the SMT solver called Z3 exhaustively [3]. Z3 can use arithmetic operations, bit vectors, arrays, and recode types. Since an SMT solver searches the answer in bounded space exhaustively, it can verify that no assignment violates the consistency rules.

4 EXAMPLE SHOWING MOTIVATION OF THIS STUDY

In this section, we present the motivation of this study by showing an example.

EQ [7] detected equals methods violating the consistency rules by experiments for four open-source projects. The class implemented equals methods which may cause a fault that is hard to detect. If an instance of a class which implements its equals method violating the consistency rules is used in the standard collection, unexpected behavior might cause faults. For example, if an instance of a class which has the equals method violating the reflexive rule is used in a standard collection, a contains method of the standard collection cannot determine correctly whether the collection contains such an instance. To check the equivalence of instances, a contains

```
int [] array1 = {3, 6, 2, 9};
int [] array2 = {3, 6, 2, 9};

COSString a = new COSString (array1);
COSString b = new COSString (array2);

// let assume that a.equals(b) but a.hashCode() != b.hashCode()

Set<COSString> database = new HashSet<COSString>();
database.add(a);
if (database.contains(b)) }
  // expect behavior
 System.out.println("Users want this.");
} else {
 // but this clause is executed
 System.out.println("However this statement is executed.");
}
```

Figure 3: Motivation Example

method of a collection such as List uses equals methods, an unexpected behavior might occur. Also, if equals methods judge two instances are equivalent but these two instances return different hash values, the hashCode methods cannot perform the correct behavior. For example, HashMap may contain two instances judged equivalent by the equals methods. Figure 2 shows an example of the motivation of this study. This example shows an implementation of the hashCode method violating the consistency rules in PDFBox of Apache [18].

PDFBox uses java.util.Arrays.equals as the equals method of the COSString class. Also, PDFBox uses the hashCode method of a byte array as the hashCode method of the COSString class. Hence, the equals method checks whether two arrays have the same number of the elements and all corresponding pairs of the elements in the two arrays are equal. The hashCode method checks whether these two arrays have the same memory address. Therefore, if instances of the arrays are different and these arrays have the same elements with the same order, the equals method judges these two objects are equivalent but the hashCode method returns a different hash value for each. In this case, HashSet may contain two instances judged equivalent by the equals methods.

It is important that both of equals method and hashCode method are rightly implemented, because incorrect implementation will causes unwanted behavior when programmer uses Java Collection Frame Work with it.

For example, let us consider the program in Fig. 3. Let assume that `a.equals(b)` holds but `a.hashCode() != b.hashCode()` also holds, in other words, we implement incorrectly hashCode method against its equals method. The program in Fig. 3 executes `else` clause which we do not expect.

Thus, it is important that we check whether both of equals method and hashCode method to be rightly implemented. The paper (EQ [7]) already has proposed a method for checking equals method, and therefore we focus on hashCode method.

To avoid such unexpected behavior, we propose a new method that checks whether both equals and hashCode methods satisfy the consistency rules.

Note that the implementation of both of equals method and hashCode method is programmers' obligation regardless the

```
public class ArEntry implements ArConstants{
   private String filename;
   public String getFilename() {
      return this.filename;
   }
   public boolean equals(Object it) {
      if (it == null || getClass() != it.getClass())
         return false;
   return equals((ArEntry) it);
   }
   public boolean equals(ArEntry it)
   if (this.filename == null)
      return (it.getfilename() == null);
   else
      return
         this.getFilename().equals(it.getFilename());
   }
   public int hashCode() {
      return super.hashCode();
   }
}
```

Figure 4: Conversion example of Java source code

version of Java.

5 OUR PROPOSED METHOD

Our proposed method analyzes the Java code and models the behavior of both the equals and the hashCode methods in the model description language called SMT-LIB. The model is checked by Z3. Our proposed method receives the type hierarchy of the code and then outputs whether each equals method satisfies the consistency rules. The proposed method is based on static analysis.

Our proposed method consists of the following four steps. 1) It performs path analysis for the equals method. 2) It analyzes the pattern of the equals method. 3) It converts a given Java code to a model described in SMT-LIB. 4) It verifies the model by Z3. The path analysis generates a control flow graph and performs data flow analysis. The data flow analysis specifies what class is referred by a reference variable at each position of the source code and specifies what methods are called. Then, specified methods are inlined into equals or hashCode methods if needed. The equals or hashCode methods perform some types of procedures. Therefore, pattern analysis classifies each method into some patterns. Because it is difficult to directly convert the hashCode procedures which contain loops including arithmetic operation or library calls, we analyze this procedure by using heuristic operations. After pattern analysis, we convert Java code to SMT-LIB based on information from the pattern analysis. Also, to check for violations of the obtained consistency rules, we give some constraints to the SMT-LIB model. It is very difficult to model the first consistency rule of the hashCode method. It should be recalled that the rule is "Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application." To model this rule, it is necessary to model the concept of time. However, since first-order logic cannot represent the concept of time, an SMT solver cannot check the first consistency rule of the hashCode methods. Therefore, to resolve this problem, we introduce a more strict consistency rule which replaces the first hashCode rule. On the other hand, since the second consistency rule of the hashCode methods is representable in first-order logic, an SMT solver can check the second consistency rule of the hashCode methods directly. The substituted consistency rule of the hashCode method is as follows. We define the first rule below as the subset rule and the second one as the equivalence rule.

- Subset rule: Set of fields used in hashCode methods must be subsumed by the set of fields used in equals methods.
- Equivalence rule: If two objects are equal according to the `equals(Object)` method, then calling the hashCode method on each of the two objects must produce the same integer result.

As Equivalence rule, Java specification gives a one-way rule. Therefore the rule (1)"`a.equals(b)` then `a.hashCode() == b.hashCode()`" is necessary while(1') "`a.hashCode() == b.hashCode()` impies `a.equals(b)`" does not need to be held. Our proposed method uses only (1) as Equivalence rule.

Figures 4 and 5 show examples of converting Java source code (Fig. 4) to a model written by SMT-LIB (Fig. 5). In this example, the type hierarchy has three classes. That is, the classes are an ArConstants interface, ArEntry class which implements ArConstants and overrides equals and hashCode methods, and a class implementing ArConstants that does not override equals and hashCode methods (this class is represented as UnderARC in Fig. 5). Figure 5 represents the SMT-LIB model of the source code in the type hierarchy. Figure 5 represents the declaration of types by the class information, a definition of the method behavior by the method information, and the constraints used for validation by an equality check.

5.1 Path Analysis

The path analysis is similar to that of [7]. First, our method searches equals and hashCode methods. Our method traces the inheritance relation for a class which does not override its equals and hashCode methods. If we detect the class which overrides the equals and hashCode methods, we regard the equals and hashCode methods of its parent class as the equals and hashCode methods of such a class. If no overrides of the equals and hashCode methods are found in an inheritance relation, we regard the equals and hashCode methods of Object class as the equals and hashCode in such a class. Next, we analyze Java byte code using Soot [19] and generate its control

```
;Class information
(declare-datatypes () ((Type ArEntry ArConstants UnderARC Object Null)) )
...
(declare-datatypes () (( Ref(Rfield (eqnum Int) (hsnum Int) (pointer Int)) )) )
(declare-datatypes () ((ArEntry(Arfield (filename Ref)) )) )
(declare-datatypes () (( Object(Ofield (ar ArEntry)(pointer Int)(class Type)))))
(declare-const this Object)
(declare-const that Object)
(declare-const other Object)
(declare-const nobj Object)
...
;method information
(define-fun equalsRef ((r1 Ref)(r2 Ref)) Bool
   (ite (and (and (not (= (pointer r1) 0)) (not (= (pointer r2) 0))) (= (eqnum r1)(eqnum r2))) true false ))
(define-fun equalsMain ((o1 Object)(o2 Object)) Bool
   (and (=> (or (= (class o1) ArConstants) (or (= (class o1) UnderARC)(= (class o1) Object)))
         (= (pointer o1)(pointer o2)))
         (=> (= (class o1) ArEntry) (and (and (not(= (pointer o2) 0)) (= (class o1)(class o2)))
         (or (and (= (pointer(filename (ar o1))) 0) (= (pointer(filename (aro2))) 0))
         (equalsRef (filename (ar o1)) (filename (ar o2)))) ) )
   )
)
(define-fun hashCode ((o1 Object)) Int(pointer o1))
;equality check
...
(assert (not (equalsMain this this) ) )
...
(assert (not(iff (equalsMain this that) (equalsMain that this)) ) )
...
(assert (not(=> (and (equalsMain this that) (equalsMain that other))
(equalsMain this other)) ) )
...
(assert (not(=> (not(= (pointer this) 0)) (not(equalsMain this nobj))) ) )
...
;hashCode check
(assert (not(=> (equalsMain this that) (= (hashCode this) (hashCode that) )) ) )
...
```

Figure 5: Conversion example of SMT-LIB

flow graph. This control flow graph is represented by Jimple. Jimple represents a Java source code as a three-address code, in which each expression consists of one operator, two operands, and one variable which stores the result of the operation. Hereafter, we analyze a Jimple code generated by Soot.

Our method performs a path analysis. First, our method enumerates paths by using the obtained control flow graph. Next, our method performs a data flow analysis for each path, and specifies what class is referred from a reference variable at each source code location and what methods are called. With this information, our method performs inlining of the method invocations in equals or hashCode methods. However, since the number of method invocations is very large, our method limits the inlining. Our method inlines the method invocations only in the type hierarchy. Also, our method does not inline a getter method, which is modeled as directly referring the field values. Although our method does not inline outer methods, it models methods of Object class, wrapper classes, Array classes, and Collections, because the behaviors of these methods are already well known.

Finally, our method trims the path which is unreachable and not necessary to our model. Since our method models the equals method as returning true, we trim the path which returns false. Also, to avoid modeling the null pointer exception, our method trims the path which includes uninitialized reference variables. In other words, our method enhances the performance by trimming the path not necessary to a model.

5.2 Analyzing the Pattern of Methods

In this step, our method analyzes the pattern of the procedure in equals and hashCode methods. By referring to the modeling rules for each pattern, our method converts Java source code to SMT-LIB. In addition to the pattern analysis,

our method checks whether a subset rule is violated in this step.

5.2.1 Analyzing Patterns of equals Methods

EQ introduce the six pattern of procedures in equals methods. Our method analyzes what pattern matches the equals methods. The six procedure patterns are equivalence checking of array, equivalence checking of List, equivalence checking of Set, equivalence checking of Map, type checking, and state checking. Type checking looks for the existence of the following: checking by an instance operator in an if expression, typecasting by a cast operator, type checking by getClass method in Object class. State checking looks for the existence of the equivalence checking of field values and checking a reference variable that is not null. Equivalence checking of Array, List, Set, and Map checks whether elements in each structure can be compared by a loop.

5.2.2 Analyzing Patterns of hashCode Methods

We introduce the pattern of the procedure of hashCode methods and define the rules of each procedure. The hashCode method procedure patterns are converting to int, a bit operation, and an arithmetic operation in loop. Converting to int checks for the existence of type converting by cast operation and type converting by library method of wrapper class. The arithmetic operation in loop checks the existence of the procedure of an add operation in loop.

5.2.3 Checking of the Subset Rule

Our method performs checking of the subset rule. Our method collects a set of field variables used in equals and hashCode methods by analyzing the equals method and the hashCode methods, and checks whether the set of field variables used in hashCode methods are subsumed by the set of field variables used in equals methods. If a hashCode method invokes the method of the parent classes and other methods, since the path analysis inlines the method of the parent classes and other methods in hashCode methods, the set of field variables used in the hashCode method contains field variables used in such method. If the values of variables in the method of parent classes and other methods are changed, the change affects the return value of equals and hashCode methods. Therefore, since it is necessary to consider such field values, we substitute a subset rule for the first rules of hashCode methods. Two cases occur in the consistence rule of hashCode methods. One is that hashCode methods use fields values used in the equals method. In this case, if field values used in the equals method are not changed, hash values also do not change. The other case is that hashCode methods use not only field values used in the equals method but also field values not used in equals methods. In this case, nevertheless, the field values used in the equals method do not change, but hash values possibly change. To check this case, it is necessary to check relations of field values used in equals and hashCode methods. Since it is necessary to check all methods which modify field values, analyzing consumes many resources.

```
(declare-datatypes () ((Type ArEntry ArConstants UnderARC
  Object Null)) )
(define-fun subof ((t1 Type) (t2 Type)) Bool
  (ite (or (= t1 Null) (= t2 Null)) false
    (ite (and (= t1 ArEntry) (= t2 ArConstants)) true
      (ite (and (= t1 UnderARC) (= t2 ArConstants)) true
        false
      )
    )
  )
)
(declare-fun instanceof (Type Type) Bool)
(assert (forall ((x Type) (y Type))
  (=> (subof x y) (instanceof x y))))
(assert (forall ((x Type) (y Type))
  (=> (and (instanceof x y) (instanceof y x))
    (= x y))))
(assert (forall ((x Type) (y Type) (z Type))
  (=> (and (instanceof x y) (instanceof y z))
    (instanceof x z))))
(assert (forall ((x Type)) (= (instanceof Null x) false) ))
(assert (forall ((x Type)) (=> (not(= x Null)) (instanceof x
  Object) )))
(assert (forall ((x Type)) (=> (not(= x Null)) (instanceof x x) )))
(assert (forall ((x Type)) (=> (not(= x ArEntry)) (not(instanceof x
  ArEntry)) )))
(assert (forall ((x Type)) (=> (not(= x UnderARC))
(not(instanceof x UnderARC)) )))
```

Figure 6: Model of the instanceof operation

5.3 Conversion of Java Source Code to SMT-LIB

This step consists of the following two steps. 1) The basic structure conversion converts methods, inheritance relations, classes, and field values to SMT-LIB. 2) The procedure of the method conversion converts the procedure of the method to SMTLIB by using information obtained from the step of analyzing the pattern of methods.

5.3.1 Basic Structure Conversion

Our method represents classes and fields by records in SMT-LIB. Our method defines fields used in equals and hashCode methods. It converts all primitive values to Ints in SMT-LIB. Since equals methods perform only comparisons, Int has enough power to represent the result of equivalence checking.

Since hashCode methods perform any type of arithmetic operations and usually perform typecast to int type before arithmetic operations, our method always converts primitive types used in hashCode methods to Ints. Our method converts the enumeration field to the enum type in SMT-LIB. Since reference variables of enum types possibly refer null, our method models add a NULL value to the identifier introduced by the enum type. Also, since the enum type of hashCode methods invokes a hashCode method of Object class, our methods models the enum type of hashCode methods as returning different values for each identifier. Our method defines reference type fields by introducing the new record Ref

Table 1: Some of the simple μ conversion rules

$\mu(n_1+n_2\ )$	=	+ $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1-n_2\ )$	=	- $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1*n_2\ )$	=	* $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1/n_2\ )$	=	/ $\mu(n_1)$ $\mu(n_2)$
$\mu(a_1==a_2\ )$	=	= $\mu(a_1)$ $\mu(a_2)$
$\mu(n_1<n_2\ )$	=	$<$ $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1>n_2\ )$	=	$>$ $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1>=n_2\ )$	=	$>=$ $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1<=n_2\ )$	=	$<=$ $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1!=n_2\ )$	=	not(= $\mu(n_1)$ $\mu(n_2)$)
$\mu(b_1\|\|b_2\ )$	=	or $\mu(b_1)$ $\mu(b_2)$
$\mu(b_1\&\&b_2\ )$	=	and $\mu(b_1)$ $\mu(b_2)$
$\mu(!b_1\ )$	=	not $\mu(b_1)$
$\mu(a_1 instanceof a_2\ )$	=	instanceof $\mu(a_1)$ $\mu(a_2)$
$\mu(a_1$. getClass())	=	class $\mu(a_1)$
$\mu(T_1$. class)	=	$\mu(T_1)$
$\mu(b_1?a_1{:}a_2)$	=	ite ($\mu(b_1)$) ($\mu(a_1)$) ($\mu(a_1)$)
$\mu(n_1\|n_2\ )$	=	bvor $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1\&n_2\ )$	=	bvand $\mu(n_1)$ $\mu(n_2)$
$\mu(n_1\ \hat{}\ n_2\ )$	=	bvxor $\mu(n_1)$ $\mu(n_2)$

representing a reference type. Ref represents the object that is out of the type hierarchy. Our method models such an object based on the hypothesis that such a method satisfies the consistency rules of equals and hashCode methods. Ref defines a field variable that represents the reference of its object. It is used in equivalence checking as the Int type field. Our method defines the equals methods of Ref when a Ref object is used. Our method does not define hashCode methods of Ref. It models this as a reference of the hash values. Our method models the data structure of Java by arrays and lists. Our method represents arrays, Sets, and Maps using arrays of SMT-LIB. An array of SMT-LIB is defined by specifying the type of its index and its type of elements. For example, specifying the type of index as Int represents the array. Set is also represented by adding a constraint in which elements are different from each other in this array. Our method represents the inheritance relation of a class by the nest of records. However, it cannot model the behavior of instanceof, which checks whether a class has an inheritance relation between other classes. Hence, our method introduces the type named Type which enumerates the type of adds null to all classes in the type hierarchy. Our method models the instanceof operator by representing the relation of Type. Figure 6 shows an example of an instanceof operation model. The definition of Object class defines all classes as a field. Object class represents the runtime objects and defines a pointer as an Int type. Type defines a field representing where the instance comes from.

5.3.2 Conversion of the Procedure of Methods

Conversion of the procedure of methods converts Java source code to SMT-LIB based on information obtained from the step of analyzing the pattern of methods. First, our method generates expression trees for each expression represented as Jimple. Our method specifies the final expression returned by the return expression by tracing the expression tree and analyzing how the values of variables are calculated. The operation in expressions is converted by the converting rules. Table 1 shows the simple converting rules of Java source code to SMT-LIB. The convert function converts Java source code to SMT-LIB, where bm and am represent subexpressions of the boolean type and the numerical type, respectively. Tm represents arbitrary types. Java represents an expression with infix notation, whereas SMT-LIB represents expressions by prefix notation. Also, our method converts the instanceof operator based on modeling previously described.

5.3.3 Conversion of equals Methods

Our method converts equals methods based on the six patterns obtained from the pattern analysis. The operations used in type checking are converted as shown in Table 1. Since verification by an SMT solver is performed on the object level, cast operations used in the equals method are not converted. Since statement checking compares values, the comparison expression is converted as in Table 1. With regard to the equivalence checking of arrays, Lists, Sets, and Maps, our method models the method which performs a comparison in the loop as it performs a comparison of each element of an array. For example, let us consider an instance of a class which has the array as the field, and performs an equals method. Our method checks whether this equals method performs a comparison of its field array with the array of its argument by the same index. Next, our method checks whether a variable used in the loop header is used as the index of the array. If those two conditions are satisfied, our method determines it performs a comparison. Most loop operations in an equals method match this pattern. Since other loop operations are rarely performed and SMT-LIB cannot evaluate statements dynamically, our method does not model such loop operations.

5.3.4 Conversion of hashCode Methods

Our method converts hashCode methods based on the six patterns obtained from the pattern analysis. A variable changed by its type by a cast operation or a method of the Java class library is represented as the Int type of SMT-LIB. Operands of bit operations are represented as 8-bit type vector types. Conversion results of the operations to Int types are obtained by applying bv2int functions to the result. Although Int of Java is 32 bits, if it models it as 32 bits, modeling takes an enormous amount of time. Therefore, our method models it as an 8-bit integer. Bit operations of hashCode methods operate two operands and do not performs bit operations on one specific bit. Hence, our method can perform verification. Arithmetic operations in a loop are analyzed and our method determines what pattern matches the operations. An arithmetic operation in a loop can be represented as expression, if the number of iterations is identical to the length of the array and arithmetic operations performed in loop do not contain nondeterministic values. However, the result of this operation is decided after the loop is terminated. Therefore, our method limits the loop iteration. This is well used in bounded model checking. Our

```
unsat
  (error "line 74 column 17: model is not available")
unsat
  (error "line 80 column 22: model is not available")
unsat
  (error "line 86 column 28: model is not available")
unsat
  (error "line 92 column 22: model is not available")
sat
  ((this (Ofield (Arfield (Rfield 8 9 7)) 3 ArEntry))
  (that (Ofield (Arfield (Rfield 8 9 10)) 2 ArEntry)))
```

Figure 7: Results of verifying the code of Figure 5

method calculates the result of the loop after 0 to 10 iterations. Our method cannot verify all cases but if our method decides a hashCode method violates the rule, this decision is absolutely true. Similarly to the equals method, our method does not model other loop operations.

5.3.5 Additional Constraints

Our method verifies the four consistency rules of equals methods and the equivalence rule of hashCode methods by an SMT solver. The SMT solver solves the constraint and shows the assignment, which is a set of values for the variables that satisfies all constraints. Therefore, to obtain an example of a type hierarchy which violates the consistency rule, our method introduces the negation of consistency rules as the constraints.

5.4 Solving Constraints by an SMT Solver

Our method verifies the SMT-LIB expression which models Java source code by using an SMT solver called Z3. In general, Z3 determines whether a given set of constraints is satisfiable. If it is unsatisfiable, Z3 also outputs a counter example, which is a set of assignments of variables and interpretation of functions.

Since our method uses the negation of the consistency rules as the constraints in SMT-LIB, if Z3 outputs unsatisfiable, then we conclude that the source code does not violate the consistency rules. On the other hand, if Z3 outputs satisfiable, we conclude that the source code violates the consistency rules In such a case, Z3 can output a set of assignments which makes the input true.

Figure 7 shows the results of verification by Z3 for the source code in Fig. 5. The bottom line shows that the result of verifying the equivalence rule of the hashCode method and the other four lines are the results of verifying the consistency rules of equals method. Figure 7 shows that violation of the equivalence rule is detected. The optional outputs as assignments show that two ArEntry objects have the same field values but their references are different.

6 EXPERIMENTS

In this section, we evaluate our proposed method by experiment. We implement the verification function of the subset

```
public class HCatFieldSchema implements Serializable {
   public enum Category {
      PRIMITIVE,ARRAY,MAP,STRUCT
   };
   String fieldName,typeString;
   Category category ;
   ...
   public boolean equals(Object obj) {
      if (this == obj)
         return true;
      if (obj == null)
         return false;
      if (!(obj instanceof HCatFieldSchema))
         return false;
      HCatFieldSchema other = (HCatFieldSchema) obj;
      if (category != other.category)
         return false;
      if (fieldName == null) {
         if (other.fieldName != null) {
            return false;
         }
      } else if (!fieldName.equals(other.fieldName)) {
          return false;
   }
      if (this.getTypeString() == null) {
         if (other.getTypeString() != null) {
            return false;
         }
      } else if (!this.getTypeString().equals(other.getTypeString())) {
         return false;
      }
         return true;
   }
   public int hashCode() {
      int result = 17;
      result = 31 * result + (category == null ? 0 : category.hashCode());
      result = 31 * result + (fieldName == null ? 0 : fieldName.hashCode());
      result = 31 * result + (getTypeString() == null ? 0 :
      getTypeString().hashCode());
      return result;
   }
}
```

Figure 8: A fixed HCatFieldSchema class

rule, part of the modeling to SMT-LIB and the verification function of our tool. We did not implement the converting of bit operations and loops. This is some of our future work. Subsection 6.1 shows the results of applying our tool to some projects. The results show the effect of methods violating the subset rule. Subsection 6.2 shows the results for whether our tool can detect violation of the consistency rules of equals methods. In the experiments, we first converted Java source code to SMT-LIB manually. Then, we applied our tool to that model. Subsection 6.3 shows the execution time of our tool. Subsection 6.4 shows how often the projects violated the rules.

6.1 Evaluation of the Subset Rule

We applied our tool to Lucene 4.6.0. Table 2 shows the results. Numclass represents the number of classes in which the equals or hashCode methods are overridden. Subset represents the number of classes satisfying the subset rule. Violation represents the number of classes violating the subset rule.

We discuss the four classes that violate the subset rule. Two of the four classes contain a field variable that stores the

Table 2: Results of violation of the subset rule

Name	NumClass	Subset	Violation
Lucene	110	106	4

length of the array and is used in only hashCode methods. The length of array can be calculated by the fields variable of the array. Also, array is used in both equals and hashCode methods. Therefore, these classes do not completely violate the subset rule. Although these fields are declared with a keyword "final", our method guarantees that the reference variables refer always to the same object, but it does not guarantee that the objects are not changed. Therefore, if the length of the array changes, the field variable is not renewed and it does not store the correct value.

One of the four classes contains a field variable that stores the hash value already calculated for performance improvement. This class returns the hash value generated by converting the memory address of object to an integer value. Since this value does not change at runtime of the application, the class does not completely violate the subset rule.

The last class does not override its equals method and invokes the equals method of Object class. The equals method of Object class does not use field values. However, this class overrides its hashCode method and uses a field value. Therefore, this class violates the subset rule.

6.2 Evaluation of the Equivalence Rule

We evaluated the equivalence rule through the HcatFieldSchema class of Apache Hive. This class receives a bug report which states that the class overrides its equals method but does not override its hashCode method in the past revision. This bug is fixed in the later revision. We manually modeled the two revisions of this class. One contains the bug and the other fixes the bug. We conclude that our tool works correctly, if the following two conditions are satisfied. 1) Our tool detects that an unfixed class violates the consistency rules. 2) Our tool detects that a fixed class does not violate the consistency rules. Figure 6 shows the source code of the fixed class. This class does not have its parent class. The unfixed class does not override its hashCode method. If the hashCode method of the unfixed class is invoked, the unfixed class invokes the hashCode method of Object. The equals method of this class determines the equivalence of two objects by comparing field values. However, the hashCode method returns true if two objects are the same. Hence, this class violates the equivalence rule. Since the hashCode method of the fixed class returns a hash value by performing arithmetic operations involving a field value used in the equals method, the fixed class does not violate the equivalence rule. We check the violation of the equivalence rule by Z3. Z3 determines the unfixed class violates the equivalence rule, but the fixed class does not violate the equivalence rule. This result shows that our method can detect the implementation which violates the equivalence rule.

Table 3: Comparison of execution times

Name	Path length	Path analysis	Pattern procedure	analysis	Execution time
Lucene	16,970	12s	29s	1s	48s
Tomcat	257,590	38s	240s	2s	285s
JFreeChart	3,538,281	11,181s	11,491s	6s	22,689s

Table 4: Number of violated rules

Name	equals method				hashCode method		total
	reflexive	symmetric	transitive	null	subset	equivalence	
Lucene	2	0	0	0	4	1	7
Tomcat	11	3	4	3	14	7	35
JFreeChart	1	1	2	0	76	36	113

6.3 Execution Times

To evaluate the cost of checking, we applied our tool to Lucene 4.6.0, Tomcat 8.0.1, and JFreeChart 1.0.17. We compared the execution times. Figure 3 shows the results of this experiment. The path length, the name of each step, and the time represent the total path length of each project, the execution time of each step, and the total execution time, respectively. Time represents the total execution time.

These results show that our proposed method is effective when it checks small or medium-sized projects. Our method can check large projects by limiting and reducing the search space. The execution time is approximately in proportion to the total pass length. We do not have an obvious answer to the cause of this result. Analyzing the cause is future work. Also, analyzing the procedure of a method and converting the Java source code to SMT-LIB model consume over 50% of the total execution time. We can reduce the total execution time by improving the performance of these steps.

6.4 Evaluation of Projects

We evaluated how often the projects violate the consistency rules. We applied our tool to Lucene 4.6.0, Tomcat 8.0.1, and JFreeChart 1.0.17.

Table 4 shows the results of this experiment. Each name in the rule column represents the number of implementations violating that rule.

We discuss the causes of the violations of the consistency rules. The causes of violating the rules of the equals methods are those of [7]. That is, they are asymmetry null checking, invalid type checking at type hierarchy, and mistyping. Also, we model the method invocations for fields as a nondeterministic function, and such modeling may generate wrong models. Three type hierarchies violating the rules are caused by the wrong models. This problem can be solved by improving our tool. For example, we can solve this problem by using the information of method behavior from users for a method which is not inlined.

Regarding the subset rule of hashCode methods, some classes contain a field variable which stores the hash value already calculated for improving the performance. This method returns the hash value generated by converting the memory address of the object to an integer value. Since this value does not change at runtime of the application, the class does not completely violate the subset rule. Also, regarding the equivalence rule, many classes override their equals methods but

do not override their hashCode methods, and so they violate this rule. This violation is only in JFreeChart, not the other two projects. Therefore, the policy of implementation of the project may affect this result. Consequently, we claim that the projects policy must contain the rule that if a class overrides the equals methods, then the class must override the hashCode methods. Also, two classes violate the equivalence rule of the hashCode methods. This violation is caused by their equals methods that violate the consistency rules.

7 CONCLUSION

In this paper, we proposed a method that verifies the consistency between both equals and hashCode methods. Also, we evaluated our method by experiments. Our method analyzes Java source code and converts the code to SMT-LIB. By using Z3, our method verifies whether the source code violates the consistency rules. If thee code violates any of the consistency rules, our method is able to output counter examples. The experimental results show that our method detects that some of the real code includes incorrect method implementations which violate some of the consistency rules.

We will implement the functions which are not yet implemented in our tool. Also, we will evaluate the performance of our tool by applying our tool to many practical projects. Experimental results show that our method detects the inconsistency of some projects, but does not show how many projects can be checked by our tool. We will apply our method to many projects and examine the results. These are all future work.

REFERENCES

[1] J. Bloch, "Effective Java," Addison-Wesley (2008).
[2] Oracle, "Java Platform, Standard Edition 7 API Specification" (2013) http://docs.oracle.com/javase/7/docs/api/.
[3] D. Hovemeyer and W. Pugh, "Finding bugs is easy," ACM SIGPLAN Notices Homepage archive, pp.92-106 (2004).
[4] M. Vaziri, F. Tip, S. Fink, and J. Dolby, "Declarative Object Identity Using Relation Types," Proceedings of the21st European Conference on Object-Oriented Programming, pp.54-78 (2007).
[5] C. R. Rupakheti and D. Hou, "An Empirical Study of the Design and Implementation of Object Equality in Java,"' Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds, pp.111-125 (2008).
[6] C. R. Rupakheti and D. Hou, "An Abstraction-Oriented, Path-Based Approach for Analyzing Object Equality in Java," Proceedings of the 17th Working Conference on Reverse Engineering, pp.205-214 (2010).
[7] C. R. Rupakheti and D. Hou, "Finding Errors from Reverse Engineered Equality Models using a Constraint Solver," Proceedings of the 28th IEEE International Conference on Software Maintenance, pp.77-86 (2012).
[8] L. deMoura and N. Bjorner, "Z3: An Efficient SMT Solver," Proceedings of the 14th international conference on Tools and algorithms for the construction and analysis of systems, pp.337-340 (2008).
[9] C. Barrett, A. Stump and C. Tinelli, "The SMT-LIB Standard Version 2.0" (2010).
[10] D. Rayside, Z. Benjamin, R. Singh, J.P. Near, A. Milicevic, and D. Jackson, "Equality and Hashing for (almost) Free: Generating Implementations from Abstraction Functions," Proceedings of the 31st International Conference on Software Engineering, pp.342-352 (2009).
[11] N. Grech, J. Rathke, and B. Fischer, "JEqualityGen: Generating Equality and Hashing Methods," Proceedings of the 9th international conference on Generative programming and component engineering, pp.177-186 (2010).
[12] T. Jensen, F. Kirchner, and D. Pichardie, "Secure the clones: Static enforcement of policies for secure object copying," Proceedings of the 20th European conference on Programming languages and systems: part of the joint European conferences on theory and practice of software, pp.317-337 (2010).
[13] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A Challenging Model Transformation," Proceedings of the ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems, pp.436-450 (2007).
[14] T. Liu, M. Nagel, and M. Taghdiri, "Bounded Program Verification using an SMT Solver: A Case Study," Proceedings of the 5th International Conference on Software Testing, Verification and Validation, pp.101-110 (2012).
[15] I. P. Gent, C. Jefferson, and I. Miguel, "Minion: A Fast, Scalable, Constraint Solver," Proceedings of the 17th European Conference on Artificial Intelligence, pp.98-102 (2006).
[16] D. Balasubramaniam, C. Jefferson, L. Kotthoff, I. Miguel, and P. Nightingale, "An Automated Approach to Generating Efficient Constraint Solvers," Proceedings of the 2012 International Conference on Software Engineering, pp.661-671 (2012).
[17] L. Burdy, Y. Cheon, D.R. Cok, M.D. Ernst, J.R. Kiniry, G. T. Leavens, K. R. M. Leino, and E. Poll, "An overview of JML tools and applications," International Journal on Software Tools for Technology Transfer, pp.212-232 (2005).
[18] Apache, "Apache PDFBox - A Java PDF Library" (2012) http://pdfbox.apache.org/.
[19] R. Vallee-Rai, L. Hendren, V. Sundaresan, P. Lam, E. Gagnon, and P. Co, "Soot a Java Optimization Framework," Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research, pp.125-135 (1999).

(Received November 20, 2014)
(Revised Feburary 23, 2015)

Kozo Okano received his BE, ME, and PhD degrees in Information and Computer Sciences from Osaka University in 1990, 1992, and 1995, respectively. From 2002 to 2015, he was an Associate Professor at the Graduate School of Information Science and Technology of Osaka University. In 2002 and 2003, he was a visiting researcher at the Department of Computer Science of the University of Kent in Canterbury, and a visiting lecturer at the School of Computer Science of the University of Birmingham, respectively. Since 2015, he has been an Associate Professor at Department of Computer Science and Engineering, Shinshu University. His current research interests include formal methods for software and information system design. He is a member of IEEE, IEICE, IPSJ.

Hiroaki Shimba received her BI and MI degrees from Osaka University in 2012 and 2014, respectively. His research interests include model driven software development, and consistency checking between equals and hashCode Methods in Java. He now works at Fuji Xerox Corp.

Takafumi Ohta received his BI and MI degrees from Tohoku University in 2013 and from Osaka University in 2015, respectively. His research interests include bug identification using concolic execution. He now works at NS Solutions Corporation.

Hiroki Onoue received her BI from Osaka University in 2014. His theme for the bachelor degree is “ implementation of consistency checking between equals and hashCode Methods in Java. ”He now works at Sharp Corp.

Shinji Kusumoto received his BE, ME, and DE degrees in Information and Computer Sciences from Osaka University in 1988, 1990, and 1993, respectively. He is currently a Professor at the Graduate School of Information Science and Technology of Osaka University. His research interests include software metrics and software quality assurance techniques. He is a member of the IEEE, the IEEE Computer Society, IPSJ, IEICE, and JFPUG.

Proposal for Knowledge Model Using RDF-based Service Control for Balancing Security and Privacy in Ubiquitous Sensor Networks

Makoto Sato[*], Yoshimi Teshigawara[**], and Ryoichi Sasaki[*,**]

[*]Graduate School of Advanced Science and Technology, Tokyo Denki University, Japan
[**]Cyber Security Laboratory, The Research Institute of Science and Technology, Tokyo Denki University, Japan
{sato_m, teshiga}@isl.im.dendai.ac.jp, sasaki@im.dendai.ac.jp

Abstract -In ubiquitous sensor networks, various sensors and tag readers automatically collect information in space and relevant information is acquired. Efficient utilization of the acquired information is important for providing high-quality services that meet the users' privacy requirements. We use RDF triples that represent spatial information at the granularity of the requested security levels. In earlier work, we created a knowledge model that considers privacy by representing user information hierarchically, and we verified its feasibility by a simulator that we developed. Then, we extended this knowledge model. In this paper, we discuss our newly proposed extended knowledge model and its applicability to various spaces. In addition, we evaluate the feasibility of the model by using a test simulator that we developed.

Keywords: Security and Privacy, Knowledge Model, RDF, Semantic Sensor Network Ontology, Sensor Network.

1 INTRODUCTION

In ubiquitous sensor networks, various sensors and tag readers automatically collect information in space and relevant information is acquired. It is expected that the amount of information in the sensor network space will further increase due to advances of these networks. It is also expected that personal information on users will be presented with various levels of granularity. For example, GPS can acquire rough location information, and cameras can acquire detailed location information. Efficient utilization of the acquired information is important for providing high-quality services that meet the users' privacy requirements. In this regard, it is possible to identify users' personal information by combining sensor information with user information that seems to be trivial by itself. Therefore, the risk of an indirect violation of privacy makes it difficult to provide high-quality services, because protecting the user's privacy means limiting the information obtained. Thus, it is necessary to consider privacy and security. Privacy requires protection from a third party and meeting the user's privacy requirements. Security requires that the network does not leak information to the outside. Terminals that use privacy information must preserve confidentiality by means such as encryption or anonymity protection, and must impose security measures to prevent privacy information leakage or tampering.

Our research objective is RDF-based service control for balancing security and privacy. In this study, privacy information is defined as information about a behavior of the user that the sensor collects. In this paper, we focus on privacy protection.

We have been developing a platform that integrates all the information in a space by using the Resource Description Framework (RDF). An RDF represents information about a resource (subject, predicate, object) in the form of an RDF triple [1]. An RDF triple is represented by a graph in Fig. 1. The RDF expresses the subject of the resources associated with the object through the predicate. By combining inference rules and a set of vocabulary, it is possible to connect different types of data and to make the aggregation of over the partial sums. RDF triples are represented with the granularity of any spatial information. Therefore, service control information or privacy information is represented flexibly. For this reason, using the information efficiently to provide a flexible service requires organizing the RDF triples of the control information and the service state information of the space required by each service.

On the other hand, protecting personal information requires collecting this information with restrictions and proper control [2]. In our previous study, we discussed only the use of restrictions, because collecting restrictions is outside our work scope. Thus, we define privacy protection as follows. Services are allowed to use only intended information on users. Sensors are not permitted to collect unintended information on users.

We previously proposed a knowledge model that can be applied to a platform using RDF-based practical services [3]. We created a knowledge model, which is a set of vocabulary required for expressing services provided by the RDF and analyzing the RDF obtained at that time. Because our knowledge model considers privacy by representing user information hierarchically, we were able to control user information by adding a function that reflected user requests [4]. In addition, we verified the feasibility of the knowledge model by developing a test simulator.

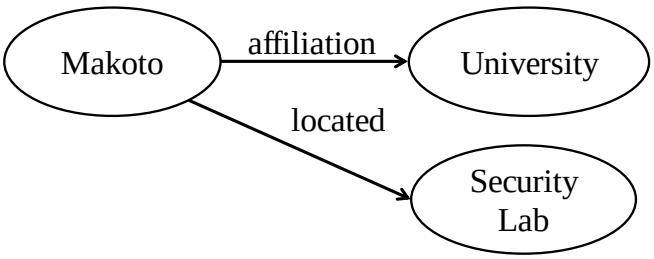


Figure 1 Example of RDF Triple

The knowledge model is represented by simple and common logic. The service provider benefits by verifying whether personal information is properly used when the service is under development. The user benefits by limiting personal information in accordance with the user's intention.

Our current work provides a level of service management for a particular space. That is, we extended the same service to a different service management system. In this paper, we discuss the applicability of our extended knowledge model to various services and various spaces, and we evaluate the feasibility of the extended model by using the test simulator.

2 RESEARCH BACKGROUND

2.1 Related Work

Various integrated management methods for sensor networks have been proposed [5]. Some of the studies represent sensor network information by using the RDF. Fujinami et al. represented a physical environment model by a location model and an object model using the RDF [6]. The location model is represented by relations between a unit space, such as a room and a building, and unit territories, such as an entrance and a kitchen. The object model is represented by object information, such as specification information and operating conditions. By using these models, developers can handle directly required information for a variety of applications. Held et al. represented user-specific information, such as user preferences, by using the RDF [7]. By evaluating context information and managing user profiles, the RDF allows for personalized, context-aware service mediation and content adaptation. Noguchi et al. managed sensor information by using the RDF to realize intelligent support systems in a room in a home [8]. In this case, the system needed a mechanism for automatically understanding information such as the sensor configurations of rooms. Therefore, they proposed an RDF sensor description to inclusively portray sensor information. It not only could describe the characteristics of the sensors, but could also easily realize an extension of the description in collaboration with other knowledge information, including new information. With these features, it allowed unified processing of sensor data. An example of the applied RDF description is the implementation of applications, such as component discovery in middleware. In our study, the service execution rules and the user requirements are centrally managed in the same way as the sensor information. Therefore, our model is expected to provide both high-quality service and protection of privacy.

Some research has discussed access control on the Web. Sacco et al. proposed the Privacy Preference Ontology that enables fine-grained access control [10]. This ontology has a vocabulary for defining fine-grained privacy preferences for RDF data. This ontology restricts a resource, a particular triple and a group of triples. By using this ontology, access control to privacy information is restricted by the properties that a requester must satisfy. Carminati et al. proposed an access control framework for social networks by specifying privacy rules using SWRL (Semantic Web Rule Language) [9].

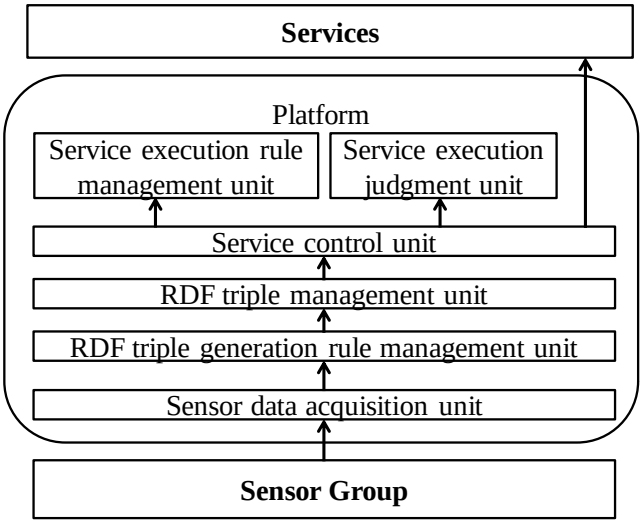


Figure 2: Overview of system functions

Additionally, user/resource relations were modeled by using RDF/OWL (Web Ontology Language). Because the Web contains a lot of privacy information, access control is effective as a method of privacy protection. Similarly, privacy protection using the RDF in a sensor network was studied. Jagtap et al. investigated privacy protection by using the RDF [11]. They proposed a model for representing the user's environment, position, and activities. An important element of their study was the use of collaborative information among sharing devices, which share and integrate knowledge about the contexts of the collaborative information. Therefore, mechanisms for privacy and security were required. They used the RDF to specify a high-level declarative policy describing the settings for sharing user information.

Our study presents a framework to provide users with an appropriate level of privacy for a mobile device and to protect the personal information gathered, including personal information that can be inferred from other information. Our study assumes an environment where the mobile devices are owned by individuals, and sensors, such as camera sensors and positioning sensors, are placed in each location. Therefore, our model is expected to protect privacy while providing a variety of services.

2.2 Development of Platform

As described in Section 1, we have been studying a method to integrate all the information in a space by describing sensor information, user information, and service states for using the RDF [12]. We aim to control services in the sensor network space by using RDF triples to provide services and information corresponding to the users' requests. Furthermore, to provide a high-quality service and to protect the privacy information of the user by reflecting the user requirements into the usage rights of RDF triples, we have been developing a platform that uses the appropriate information that satisfies the users' requirements. Figure 2 shows an overview of the functions of this platform. The functions are to generate RDF triples from the spatial

Table 1: Generated RDF triple rule

Rule	Generated RDF triple
userA is located at (x, y)	(userA, locate, (x, y))

Table 2: Service execution rule

Service	RDF triple			Excutive instruction
	Subject	Predicate	Object	
Lighting control	User	locate	Room	Light on

information acquired from sensors and to select provided services based on the RDF triples. These processes are carried out in "the RDF triple generation rule management unit" and "the service execution rule management unit". In the RDF triple generation rule management unit, RDF triples are generated from the acquired sensor information based on the RDF triples from the RDF triple generation rules. Here, the generating rule for the RDF triples is managed as a set of rules, or triggers, for generating new and already generated RDF triples. Table 1 shows an example of a generated RDF triple rule. The service execution rule management unit selects the services that can be provided by checking the RDF triples passed from the service control unit to the service execution rules. In addition, the service providing service execution rules, the RDF triples that trigger the service, and the service execution instruction are managed as a single set of rules. Table 2 shows an example of a service execution rule.

2.3 Creation of Knowledge Model

As described in Section 2.2, spatial information is represented by an RDF. We define the vocabulary and the relations of spatial information as a knowledge model expressed by the RDF. To create a knowledge model that can provide a service, it must be created after stipulating the service requirements envisioned. However, the service that runs on this platform is not yet defined. Therefore, an effective approach is to create a primary knowledge model first and then extend it gradually.

The primary knowledge model is created with a clear description of the technical issues for practical use, while considering and evaluating the services as a prototype. Specifically, the resources required for the services are assumed. Next, the state transition of the resources is expressed by an RDF graph (a set of RDF triples). Then, the resources within the RDF graph are classified into sets of the same type. A knowledge model is created to represent the relation between sets. For example, the primary knowledge model is applied to a service of the same type. A new service concept is introduced when one is lacking. Thus, by extending the knowledge model, a more general knowledge model is created.

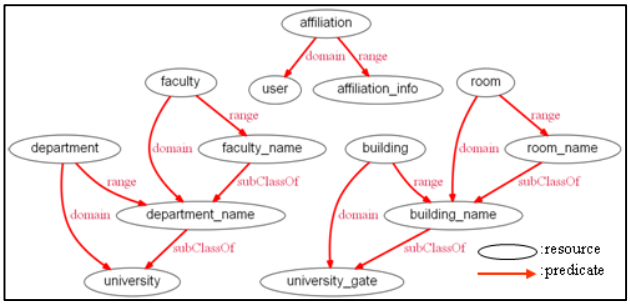


Figure 3: An example of the created knowledge model

In such a manner, we created the knowledge model shown in Fig. 3, which is intended for a university. In this figure, an ellipse represents a resource, and an arrow expresses a predicate. A feature of this knowledge model is that the domain corresponds to a subject, and the range corresponds to an object, shown as (*predicate_property*, domain, *domain_name*), (*predicate_property*, range, *range_name*). This RDF triple expresses a resource that is the subject of the relation and the object of the relation. Thus, when RDF triples are added to the RDF graph, the inference is that resources belong to a classification with a focus on the predicate [3]. In addition, by using a hierarchical representation of the affiliation information of the user, it becomes possible to restrict the use of privacy information [4]. We examined the flexibility of the service execution rules by an experiment using this knowledge model in a simulator [3].

2.4 Development of Simulator

No real system has been developed to provide services by using the knowledge model created in Section 2.3. Because the platform includes ambiguous parts, such as the storage method of the service execution rules, we cannot clearly verify the feasibility of the knowledge model. Therefore, we developed a simulator to apply the knowledge model [13].

In the simulator, we developed several functions, such as input of RDF triples, introduction of new RDF triples by inferring, reflection of user requests, and selection of executable services. Jena was used for development of the simulator [14]. Jena provides a framework for processing RDFs, and an inference engine. Graphviz was used to visualize the RDF graph [15]. We demonstrated the operation of each function and verified the feasibility of a service control by the knowledge model [4].

One of the beneficial features of the simulator is a function for reflecting user requests in order to limit the information used in the sensor network space. In Fig. 2, this function is executed in the RDF triple management unit. The user requests are managed in the form of inference rules. Specifically, RDF triples representing the restrictions (*user information*, permit, no) are added by using the inference rules, and only usable information is outputted based on these added RDF triples. Here, “no” means permit is denied.

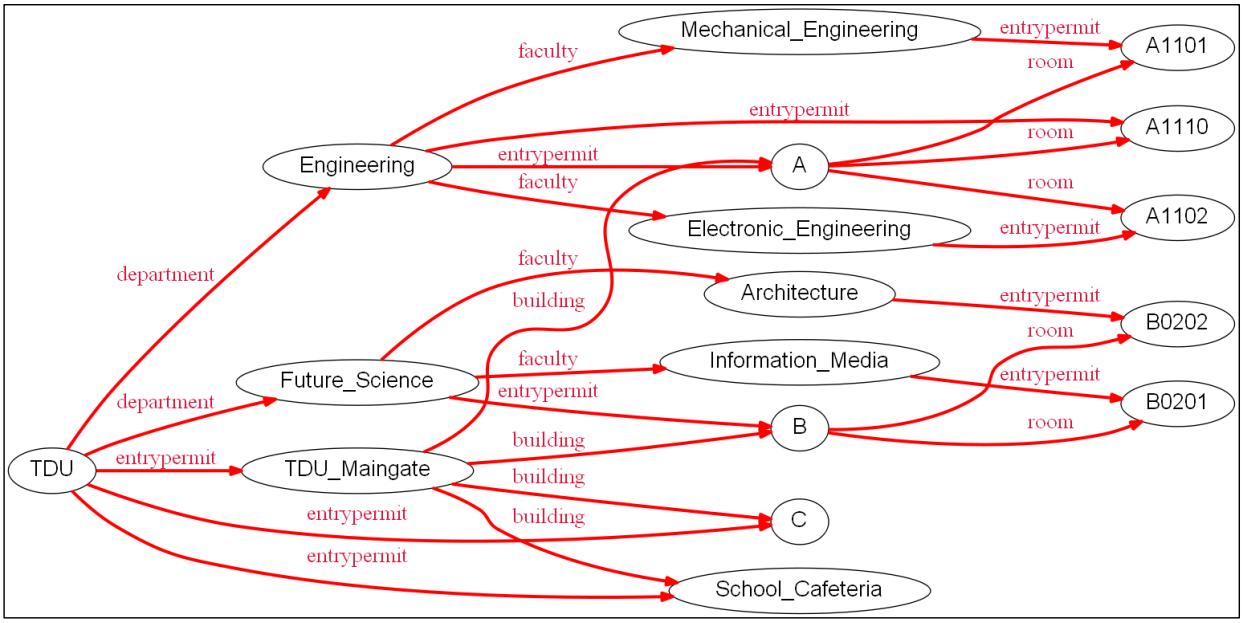


Figure 4: RDF graph for entry permit information for a faculty user

2.5 The Need for the Sensor Concept and Collecting Restrictions

The main resources in the sensor network space can be divided into space, user and service. Space is divided into "environment" and "sensor". "Environment" is a place for providing services. For example, the environment is stations, a university or a home. "Sensor" obtains the spatial information. In the previous study, we also focused on the service control using the RDF, but it was limited to only "environment" in spatial information. We considered service control information as information directly related to the service. We did not discuss the sensor at that time. Therefore, as a next step, it is necessary to incorporate the concept of the sensor to create a more general knowledge model.

In addition to introducing the concept of the sensor, it is also necessary to consider again the restrictions on collecting information, as discussed in our previous study [4]. To meet the requirements of more users, the RDF triples acquired from a sensor only use information allowed by the user. In addition, RDF triples are not applied except for those uses. The results of the study are as follows.

3 EXTENDED KNOWLEDGE MODEL

3.1 Assumed Service

The assumed service is entry management in a university campus according to the affiliation information of a user. For example, one service is unlocking the entrance door if the user is enrolled in the affiliated faculty. We make the following assumptions. The service manager is able to attach the affiliation information for the user (faculty), and the user is able to specify the affiliation information for which the service is available. We considered the "environment" as a cafeteria, three buildings, five rooms in each building and the main gate to the university campus. Figure 4 illustrates the relation between entry permission and user affiliation. A user must belong to the university in order to pass through the main gate. Similarly, the faculty must belong to the university in order to enter a building. The faculty must also belong to the corresponding department in order to enter a classroom. For example, users can enter room A1110 if they have the affiliation information of faculty. They can enter the cafeteria if they have the affiliation information of the university. Each entry has a keycard system in conjunction with the entry permit information outputted from the service, and the RDF triples are assumed to have been inputted into the system in advance by the service administrator.

Table 3: Service execution rule for room entry

Service	RDF triple			Executive instruction
	Subject	Predicate	Object	
Entry Management	User	permit	Room	Open
	User	affiliation	Affiliation information	

As described in Section 2.3, our RDF-based service extracts the information required for the service provision from the assumed service. Service execution rules are created by analyzing the information to trigger the service execution from the service contents. The service is then provided when the affiliation information for the user is inputted and room entry is allowed. For example, an RDF triple indicates whether a user can pass through the main gate of the university (university, entrypermit, maingate). In another example, the trigger for room entry is represented by the RDF triples (user, affiliation, Department of Information Media), (user, permit, Room A1110). Table 3 shows the service execution rule for room entry.

3.2 Privacy of Extended Knowledge Model

We created an extended knowledge model based on the assumed service described in Section 3.1 [4]. Specifically, the model contains the affiliation information for the user and the model of the university's sensor network space corresponding to "environment".

For the "sensor" that obtains spatial information, we used the Semantic Sensor Network Ontology (SSN) proposed by the W3C [16]. This ontology describes sensors, observations, and related concepts. For example, Sensor, Sensor Output, Sensor Input, and Device are basic resources of the sensor [17]. Therefore, we consider that these resources are sufficient as a primary model of the sensor.

It is necessary to incorporate the collecting restrictions, as described in Section 2.5. The collecting restrictions can be realized by sensors that are not permitted to collect unintended information on users. For example, a camera sensor can acquire position information, but a user who does not want to be recorded might feel that he or she wants to stop this camera sensor. If sensors are permitted to simply reflect the user's request, all sensors will be stopped. Thus, the services are likely to be provided frequently. Therefore, in terms of efficiency, we decided to stop the sensor only when all users in the space do not wish to collect the information. For this reason, to indicate the availability of the sensor, we added a predicate "hasAvailability". The predicate is the relation between the user and the sensor. In addition, a word with the prefix "ssn." indicates that it is a vocabulary of the SSN. Figure 5 shows the extended knowledge model based on this information.

Moreover, we needed to introduce new inference rules on the collecting restrictions. The collecting restrictions are realized by the following formulas using SWRL. If the user does not have the "hasAvailability" predicate, the simulator does not use the affiliation information of the user (Formula 1). If the RDF triple (*user*, hasAvailability, *sensor*) is not added, the sensor stops working (Formula 2).

$$\begin{aligned} &\text{Affiliation}(?user, ?affiliationof) \wedge \\ &\text{noValue}(?user, \text{ssh.hasAvailability}) \\ &\rightarrow \text{Permit}(?affiliationof, \text{no}) \end{aligned} \tag{1}$$

$$\begin{aligned} &\text{noValueof}(\text{ssh.hasAvailabilityBy}, ?Sensor) \\ &\rightarrow \text{Permit}(?Sensor, \text{down}) \end{aligned} \tag{2}$$

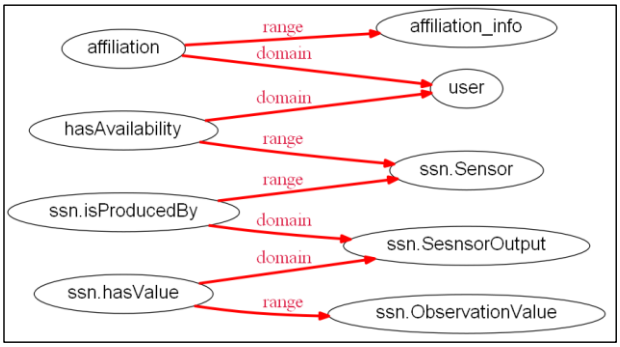


Figure 5: A part of the extended knowledge model

4 SIMULATION EXPERIMENTS

This section describes our simulation experiments. The experiments were executed to verify that the extended knowledge model is able to protect privacy information by using the user affiliation information.

4.1 Experimental Environment

The environment is the same as that described in Section 3.1. We made the following assumptions for the experiment. Each user has a smartcard. The physical location of the university is the sensor network space. The physical location can be uniquely identified by the geographical coordinates of latitude and longitude. All locations in the university have the names of identified strings assigned by public authorities. Sensors are installed in the vicinity of the door or the gate for each location. Sensors used in this space are a camera sensor and a smartcard reader. The camera sensor obtains name information for the users present in the space. The smartcard reader obtains affiliation information for the users. The individual can then be authenticated by comparing the information in the smartcard and the information acquired by the camera sensor. The acquired sensor information is converted to RDF triples automatically. The difference from the previous experiment [4] is the sensor information and increased number of relations. The relations of entry permission and affiliation information for the user are shown in Table 4.

We assumed the following two scenarios in the experiments. The difference between the two scenarios is that sensor information input is added only in scenario A (Steps 2 and 3). Then, service execution rules and the knowledge model are assumed to be stored in the database in advance. The user is assumed to be a student of Faculty of Engineering at this university in the Department of Electronic Engineering.

Scenario A:

1) Input the initial state of the assumed space. Specifically, the service administrator enters RDF triples indicating the building permit and the space information into the simulator.
2) Enter the user requirements. The user selects the available user information and the available sensors and enters them into the simulator. All sensor and user information is supposed to be available at this time.
3) The acquired sensor information, such as affiliation information, is inputted in the simulator.
4) Generate new RDF triples to perform inference processing by using the input information.
5) Determine whether the entry management service can be performed by using an RDF graph.
6) Suggest the possible entry locations.

Scenario B:

A similar scenario is carried out, but this one does not use the available sensors in Step 2 above.

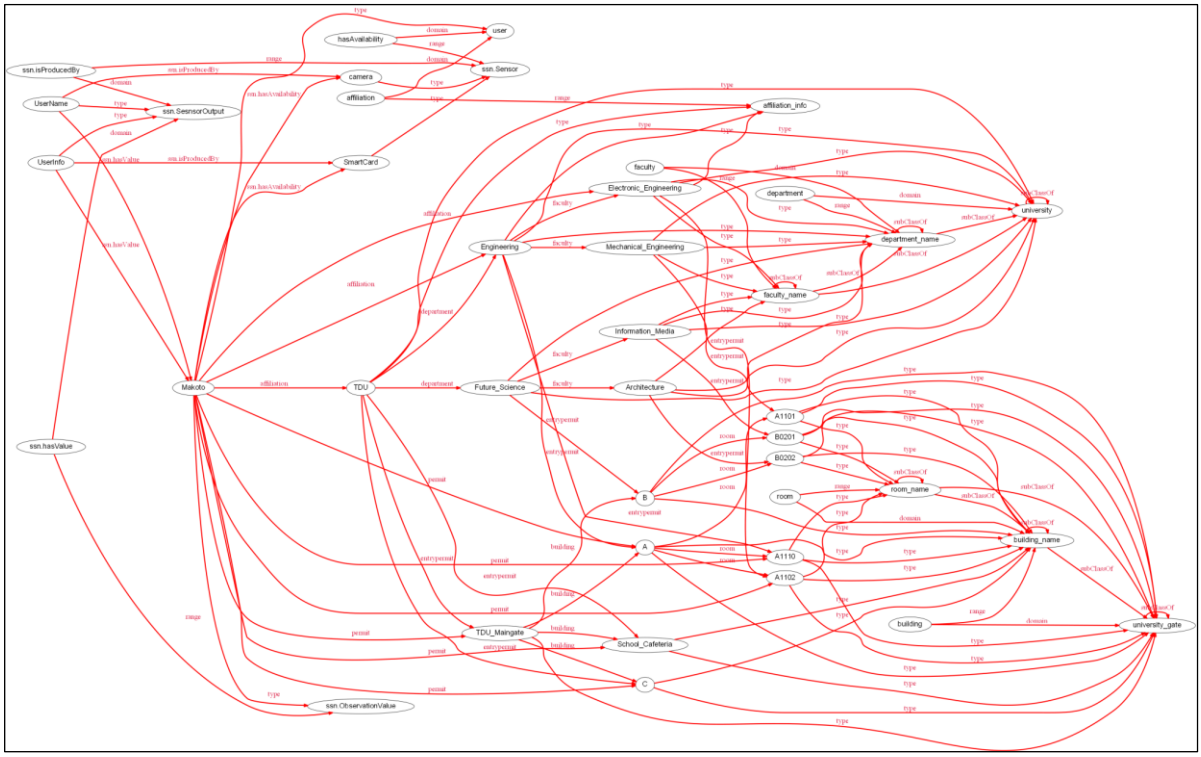


Figure 7: RDF graph after inference processing (Scenario A)

Table 4: Some of the inputted RDF triples

Subject	Predicate	Object
University	department	Engineering
University	department	Future Science
Future Science	faculty	Information Media
A	room	A1101
University	entrypermit	Maingate
Engineering	entrypermit	A
Information Media	entrypermit	A1110

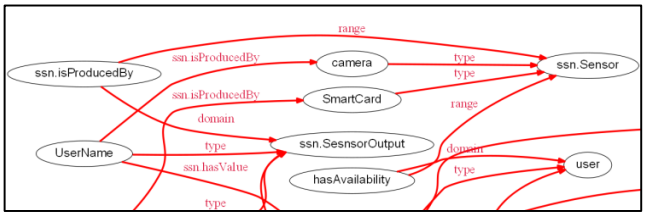


Figure 6: A part of the knowledge model

We evaluated the feasibility and flexibility of the extended knowledge model. In addition, we evaluated the feasibility of collecting restrictions to meet the user requirements.

4.2 Experimental Results

Here we present the results of the experiments described in Section 4.1, where the scenarios were executed by the simulator. Figure 6 illustrates a part of the result of Step 1. It can be seen that the smartcard and the camera sensor were associated with the SSN.

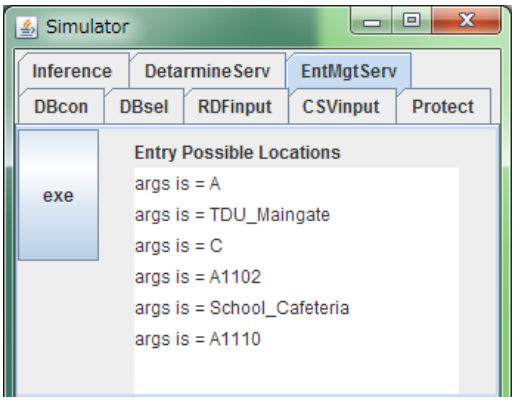


Figure 8: Screenshot of the list of locations that user is permitted to enter (Scenario A)

Figure 7 illustrates the result of Step 4 in scenario A. From this figure, it can be seen that by inference processing, many RDF triples were automatically generated. Figure 8 shows a list of the rooms where the user is permitted to enter.

Figure 9 shows the results of applying the inference rule collecting restrictions on the user request inputted in Step 2 in scenario B. The available information was eliminated because the user does not have the "hasAvailability" predicate. Figure 10 illustrates the result of Step 4. As compared with Fig. 7, Fig. 10 shows that the RDF graph is divided into two groups. This is because the affiliation information for the user is not bound to the entry permission information. Figure 11 shows a list of the rooms where the user is permitted to enter.

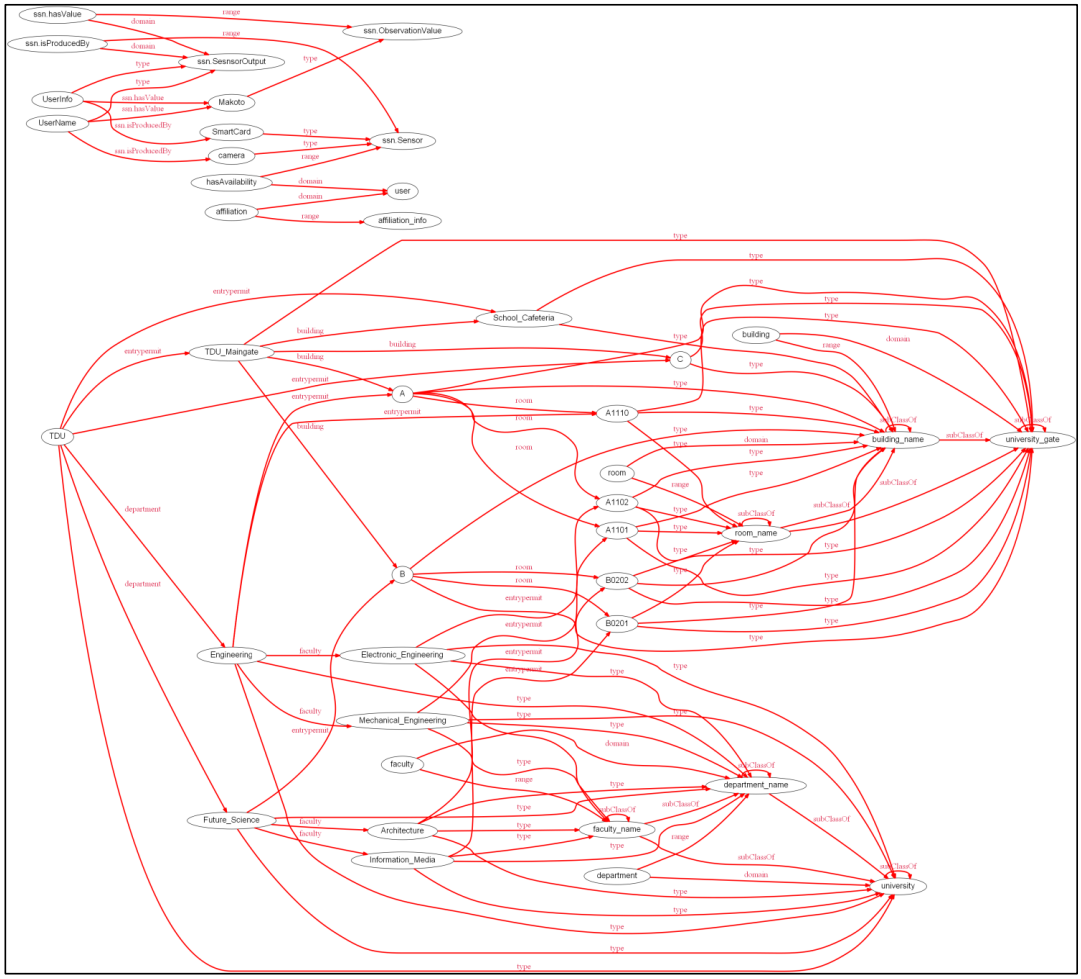


Figure 10: RDF graph after inference processing (Scenario B)

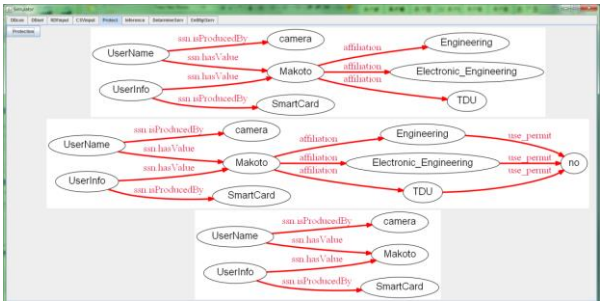


Figure 9: Screenshot of the result of collecting restrictions (Scenario B)

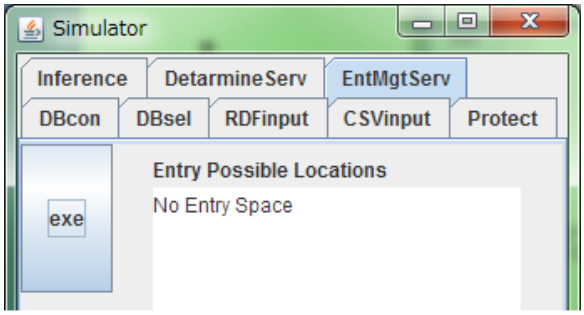


Figure 11: Screenshot of the list of rooms that the user is permitted to enter (Scenario B)

4.3 Discussion

The RDF graph in the upper part of Fig. 9 shows the user requirement. The RDF graph in the middle part of Fig. 9 can be derived from the inference rule in Formula 1 and the RDF graph in the upper part of Fig. 9. One example is (Makoto, affiliation, Electronic Engineering). The "Makoto" subject does not have the "hasAvailability" predicate. Therefore, (Electronic Engineering, use permit, no) is generated. The RDF graph in the lower part of Fig. 9 shows that the user's affiliation information was eliminated. For this reason, collecting restrictions was executed in accordance with the user by using the extended knowledge model. Moreover, the derived RDF graph shows the user affiliation information cannot be used. Therefore, it was considered that the restriction of privacy information that satisfies the users' requirements was fulfilled. However, the resource indicating the user name remained in the RDF graph of the lower part of Fig. 9. Therefore, it seems that this privacy information needs to be removed.

In Scenario A, all user information is supposed to be available. As compared with Fig. 4 and the user's affiliation information (University: TDU, department: Engineering, faculty: Electronic Engineering), all the permitted rooms are found. As a result, the entry management service lists all the permitted rooms, as shown in Fig. 8. In Scenario B, all

sensors are not supposed to be available. Thus, the permitted room does not exist. Figure 11 shows that the entry management service is not provided. For this reason, entry into permitted rooms was properly listed in the newly defined space. In the two scenarios, we confirmed that the provision of services can be automatically executed. This result shows the feasibility of the sensor network space by using the knowledge model in a variety of spaces.

The results of this study show a possible resolution to the security issue in privacy protection.

5 CONCLUSION

The purpose of this study was to confirm whether an RDF-based service implementation method keeps balance of service provisions that efficiently employ state information and privacy protection at the same time in a ubiquitous sensor network. In this paper, we extended the knowledge model by introducing the concept of a sensor by Semantic Sensor Network Ontology. In addition, we expressed the collecting restrictions by adding inference rules. We also verified the feasibility of the extended knowledge model and collecting of restrictions by experiments on the simulator that we developed. As a result, we found a possible resolution to the issue of balancing security and privacy.

REFERENCES

[1] W3C, RDF Primer (online), <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.

[2] IPA, Survey on IT Technology and Personal Information protection, IPA (2012), <http://www.ipa.go.jp/security/fy23/reports/pdata/> (*in Japanese*).

[3] M. Sato, K. Awazu, K. Kato, and Y. Teshigawara, "A Study on RDF Based Service Implementation in Ubiquitous Sensor Networks," Proc. of Multimedia Distributed Cooperative and Mobile Symposium (DICOMO2011), pp. 749-756 (2011) (*in Japanese*).

[4] M. Sato, and Y. Teshigawara, "A Proposal of a Knowledge Model in Consideration of Privacy for the RDF-based Service Control in Ubiquitous Sensor Network," Proc. Computer Security Symposium (CSS2012), pp. 246-253 (2012) (*in Japanese*).

[5] Y. Hirota, H. Kawashima, T. Umezawa, and M. Imai, "Design and Implementation of Real World Oriented Metadata Management System MeT for Semantic Sensor Network," The IEICE Transactions, Vol.J89-A, No 12, pp. 1090-1103 (2006) (*in Japanese*).

[6] K. Fujinami, and T. Nakajima, "An Information Management Infrastructure for Sentient Artefact-based Smart Spaces," IPSJ Transactions on Computing System, Vol. 47, No. SIG12(ACS 15), pp. 399-410 (2006) (*in Japanese*).

[7] A. Held, S.Buchholz, and A. Schill, "Modeling of Context Information for Pervasive Computing Applications," In Procceding of the World Multiconference on Systemics, Cybernetics and Informatics, Springer (2002).

[8] H. Noguchi, K. Tanaka, T. Mori, T. Sato, "Room Situation Search System Based on RDF Describing Room Object as Target of Human Behavior," Technical Report of IEICE, Vol. 104, No. 725, pp. 31-36 (2005) (*in Japanese*).

[9] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thurainsingham, "A Semantic Web Based Framework for Social Network Access Control," Proceedings of the 14th ACM symposium on Access control models and technologies, pp. 177-186 (2009).

[10] O. Sacco and A. Passant, "A Privacy Preference Ontology (PPO) for Linked Data," Procs of the 4th Workshop about Linked Data on the Web(LDOW-2011) (2011).

[11] P. Jagtap, A. Joshi, T. Finin, and L. Zavala, "Preserving Privacy in Context-aware Systems," 2011 Fifth IEEE International Conference, pp. 149-153 (2011).

[12] K. Awazu, D. Hirashima, K. Kato, and Y. Teshigawara, "A Study on Dynamic Space Administration and Service Control by Using RDF in Consideration of Privacy in Ubiquitous Sensor Networks," Proc. Multimedia Distributed Cooperative and Mobile Symposium (DICOMO2010), pp. 1318-1325 (2010) (*in Japanese*).

[13] M. Sato, K. Awazu, and Y. Teshigawara, "A Proposal of a Simulator for the RDF Based Service Control in Ubiquitous Sensor Networks," Proc. Multimedia Distributed Cooperative and Mobile Symposium (DICOMO2012), pp. 921-928 (2012) (*in Japanese*).

[14] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, "A. Seaborne, and K. Wilkinson, Jena: Implementing the Semantic Web Recommendations," Proc. 13th Int'l World Wide Web Conf. Alternate Track Papers and Posters, pp. 74-83 (2004).

[15] J. Ellson, E.R. Gansner, E. Koutsofios, S.C. North and G. Woodhull, "Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools," Graph Drawing Software, pp. 127-148, Springer Berlin Heidelberg (2004).

[16] W3C Semantic Sensor Network Incubator Group, Semantic Sensor Network Ontology (online), <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn >.

[17] M. Compton et al., "The SSN Ontology of the W3C Semantic Sensor Network Incubator Group," Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 17, pp. 25-32 (2012).

(Received November 20, 2014)
(Revised April 23, 2015)

Makoto Sato received his B.E. and M.E. degrees from Soka University in 2011 and 2013, respectively. He is currently doing his Ph.D. project at Tokyo Denki University. His research interests include sensor network system, ubiquitous computing and privacy issue.

Dr. Yoshimi Teshigawara is currently a senior researcher of Department of Information Systems and Multimedia Design, School of Science and Technology for Future Life as well as Cyber Security Laboratory, the Research Institute of Science and Technology at Tokyo Denki University since 2013. He began his professional career in 1970 at NEC Corporation, engaged in the design and developments of network architecture and computer systems via satellite. He worked for Soka University from 1995 to 2013, and served Dean of Faculty of Engineering and Graduate School of Engineering. His current interests are network security, e-learning, ubiquitous computing. Dr. Teshigawara received his PhD from Tokyo Institute of Technology, Japan, in 1970.

Ryoichi Sasaki is a professor of Dept. of Information Systems and Multi Media, School of Science and Technology for Future Life, Tokyo Denki University. He received his B.S. Degree in health science and Ph.D. Degree in system engineering, both from the University of Tokyo in 1971 and 1981, respectively. From April of 1971 to March of 2001, he was engaged in the research and research management on systems safety, network management and information security at Systems Development Laboratory of Hitachi Ltd. From April of 2001, he is a professor of Tokyo Denki University, and engaged in the research and education on information security. Now, he is also an advisor of Information Security in Cabinet Secretariat for Government of Japan, and a visiting professor of National Institute of Informatics, Japan.

Submission Guidance

About IJIS

International Journal of Informatics Society (ISSN 1883-4566) is published in one volume of three issues a year. One should be a member of Informatics Society for the submission of the article at least. A submission article is reviewed at least two reviewer. The online version of the journal is available at the following site: http://www.infsoc.org.

Aims and Scope of Informatics Society

The evolution of informatics heralds a new information society. It provides more convenience to our life. Informatics and technologies have been integrated by various fields. For example, mathematics, linguistics, logics, engineering, and new fields will join it. Especially, we are continuing to maintain an awareness of informatics and communication convergence. Informatics Society is the organization that tries to develop informatics and technologies with this convergence. International Journal of Informatics Society (IJIS) is the journal of Informatics Society.

Areas of interest include, but are not limited to:

Computer supported cooperative work and groupware
Intelligent transport system
Distributed Computing
Multi-media communication
Information systems
Mobile computing
Ubiquitous computing

Instruction to Authors

For detailed instructions please refer to the Authors Corner on our Web site, http://www.infsoc.org/.

Submission of manuscripts: There is no limitation of page count as full papers, each of which will be subject to a full review process. An electronic, PDF-based submission of papers is mandatory. Download and use the LaTeX2e or Microsoft Word sample IJIS formats.

http://www.infsoc.org/IJIS-Format.pdf

LaTeX2e

LaTeX2e files (ZIP) http://www.infsoc.org/template_IJIS.zip

Microsoft Word™

Sample document http://www.infsoc.org/sample_IJIS.doc

Please send the PDF file of your paper to secretariat@infsoc.org with the following information:

Title, Author: Name (Affiliation), Name (Affiliation), Corresponding Author. Address, Tel, Fax, E-mail:

Copyright

Publisher

Address: Informatics Laboratory, 3-41 Tsujimachi, Kitaku, Nagoya 462-0032, Japan
E-mail: secretariat@infsoc.org

CONTENTS