Asian Journal of Information Technology 7 (3): 117-125, 2008
ISSN: 1682-3915

Smart Health Application Implementation on UI JavaCard Based Smart Card

Riri Fitri Sari and Donny
Department of Electrical Engineering, Faculty of Engineering, University of Indonesia,
Kampus Baru UI Depok 16424, Indonesia

Abstract: Smart Card is a card with an embedded microprocessor inside the card. It can be programmed to create application, perform task and store information. Smart Card has a protocol called Application Protocol Data Unit (APDU), used to control data communication process. JavaCard is one of a technology in developing Smart Card. JavaCard is based on Java programming language to create JavaCard application. Since, the year 2006 University of Indonesia has been distributed smart cards to the new students, in order to provide a better and integrated services, using state of the art technology. The smart card is used for student's identification with biometric information, library card, attendance card, e-parking, university bike utilization, bus way utilization and other facilities such as smart health application and e-wallet in the future. Until now the smart card has been distributed to more than 40000 students. In this research, a JavaCard application called Smart Health has been developed on the University of Indonesia's Smart Card. This application has 2 functions. First, to read medical records from JavaCard and second, to write medical records into the JavaCard. Medical record which will be saved in the JavaCard was split into 6 categories. Smart Health application has 3 main parts, JavaCard applet, connector applet and terminal module. JavaCard applet is a program to control the reading and writing process on JavaCard. It also, controls JavaCard memory allocation. Connector applet is an interface program to connect the JavaCard applet with the terminal module. Terminal module is a Graphical User Interface (GUI) module. User interacts with the Smart Card application using this module. Performance analysis of this application focuses on the JavaCard memory allocation, APDU role on data traffic, data process, memory optimalization and application speed. In addition, we also analyze further development of this Smart Health application. The result of the analysis shows that JavaCard applet programming, APDU arrangement to process operation and memory allocation are the keys in building a JavaCard application.

Key words: Smart Card, JavaCard, APDU, Smart Health, medical record, memory allocation

INTRODUCTION

The main point in technology development is how to make human life simpler. Modern human life is fulfilled with many system and information. The information is stored and could be accessed in need. As information becomes more complex, there must be a procedure that the activities involved in information access can run efficiently, in terms of time, cost and security.

To reach that goal, some technology has been developed and currently being implemented in our daily life and one of them is Smart Card technology. The most appending problem in using Smart Card is how to merge ID card, ATM card, credit card and hospital card functions, filled to a single card with all important information (Surendran, 2000).

Since, the year 2006 University of Indonesia has been distributed smart cards to the new students, in order to provide a better and integrated services, using state of the art technology. The smart card is used for student's identification with biometric information, library card, attendance card, e-parking, university bike utilization, bus way utilization and other facilities such as smart health application and e-wallet in the future. Until now the smart card has been distributed to more than 40000 students. This study describes the creation of a Smart Card application for the University of Indonesia's smart card (Guarddin *et al*., 2007).

Problem definition: In this project, a Smart Card application called Smart Health has been developed. The type of Smart Card used in this application is JavaCard. JavaCard is a Java based Smart Card which uses Java programming language to create applications. Our Smart Health application works to keep the medical information and has 2 main functions, i.e., to read medical records from

Corresponding Author: Riri Fitri Sari, Department of Electrical Engineering, Faculty of Engineering, University of Indonesia, Kampus Baru UI Depok 16424, Indonesia

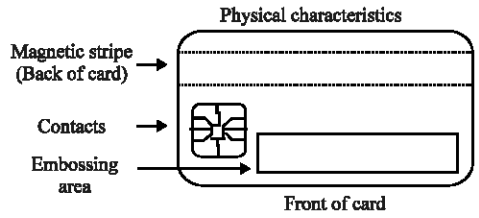

Fig. 1: Smart card physical characteristics (Chen and Giorgio, 1998)

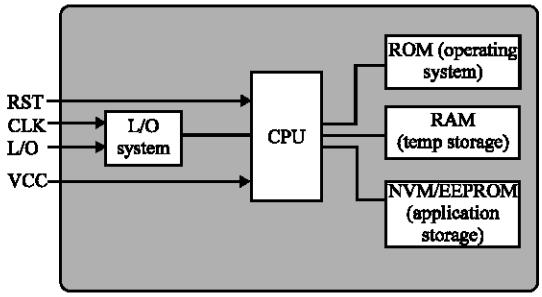

Fig. 2: Smart Card elements (Surendran, 2000)

JavaCard and to write medical records into the JavaCard. Medical record which will be saved in the JavaCard has been split into 6 categories. The application also includes personalization of the Smart Card holder. This stduy also presents the Smart Card application design and implementation, as well as the performance evaluation of the system.

Smart Card has a computer chip or microprocessor inside. This card can be programmed to do some task and store information. Card Acceptance Device (CAD), such as card reader, is needed to connect the card and a computer. There are 2 types of Smart Card, namely, Intelligent Smart Card and Memory Card. Intelligent Smart Card has the ability to read, write and calculate. Memory Card is used only to store information (Surendran, 2000). Figure 1 shows that Physical Characterisics of a smart card.

Smart Card elements: Smart Card has four main elements, Central Processing Unit (CPU), memory, input/output and Interface Device (IFD). Generally, Smart Card CPU is an 8-bit microcontroller. There are three types of memory inside Smart Card: Read Only Memory (ROM), Electrically Erasable Programmable Read Only Memory (EEPROM) and Random Access Memory (RAM). Smart Card operating system and basic software are stored in the ROM. The EEPROM is used to install and run the application. The RAM is used to perform calculation process. Figure 2 depics the elements of a smart card.

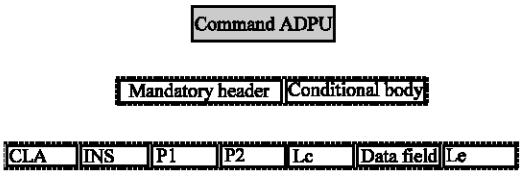

Fig. 3: Command APDU (Chen and Giorgio, 1998)

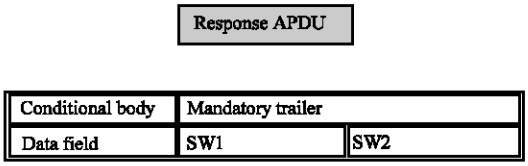

Fig. 4: Response APDU (Chen and Giorgio, 1998)

Application Protocol Data Unit (APDU): Application Protocol Data Unit (APDU) is a command message which is sent from the application layer to the Smart Card and response message being sent from the Smart Card to the application layer. Communication between Smart Card and card reader is performed using APDU message. An APDU can be considered as a packet data which contains a complete instruction or a complete response from a Smart Card. There are 2 kinds of APDU, Command APDU and Response APDU.

Smart Card always waits for a Command APDU from a terminal. It then executes the action specified in the APDU and replies to the terminal with a Response APDU. Command APDUs and Response APDUs are some information being exchanged between a card and a terminal (Di Giorgio, 1997). Figure 3 shows the command APDU, whereas Fig. 4 shows the response APDU.

JAVACARD

JavaCard is a kind of Smart Card which is capable to run Java based program. By using Java, creating Smart Card application will be much easier. The programmer can make the application using Java Card Virtual Machine (JCVM) dan Application Programming Interface (API) inside the JavaCard (Hartanto, 2007).

The JCVM is built on top of a specific Integrated Circuit (IC) and native operating system implementation. The JCVM layer hides the manufacturer's proprietary technology with a common language and system interface. The Java Card Framework defines a set of Application Programming Interface (API) classes for developing Java Card applications and for providing system services to those applications (Chen, 2000). Java

card applications are called applets. Multiple applets can reside on one card. Each applet is identified uniquely by its Application Identifier (AID) (Chen and Giorgio, 1998). Figure 5 shows tha JavaCard architecture.

SMART HEALTH APPLICATION DESIGN

JavaCard applet: JavaCard applet is a Java based program which uses javacard.framework library. This library contains Application Programming Interface (API) that supports the making of a JavaCard applet. The principal in making a JavaCard applet is how to control the APDU traffic for reading and writing process in the JavaCard. All operation inside JavaCard is performed using APDU (Di Giorgio, 1998).

JavaCard applet has one class named PKMApplet with some methods: InitializePIN(), install(), process(), setData(), sendData(), resetData(), getRecordLength(), sendAddData() and setAddData() (Chen, 1999).

These methods are used to control the application operation. The operation includes installation process, PIN arrangement for data security, reading and writing process between the terminal and JavaCard and the error handling of the application (Surendran, 2000).

First process in the JavaCard applet is the installation and selecting applet. If the user wants to read medical records stored in the card, the applet will take the data from the memory. If the user wants to write medical records, the PIN must be initialized to start the writing session. Then the writing operation can be done. After that, the session will be closed. Reading and writing process for the personal data use the same mechanism (Fodor and Hassler, 2005). Figure 6 shows the Java Card Applet Activity Diagram.

Connector applet: Connector applet is an applet that used to connect the JavaCard applet with the terminal module. Connector applet has one class named PKMApplet Connector and filled with some methods: openWrite Session(), closeWriteSession(), getMedical Record(), set MedicalRecord(), getRecord Length(), get AddData(), setAddData() and close Applet(). These methods will be the connector of the input and output traffic between the terminal and JavaCard.

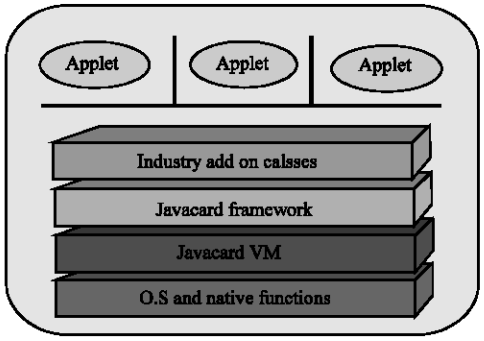

Fig. 5: JavaCard architecture (Chen and Di Giorgio, 1998)

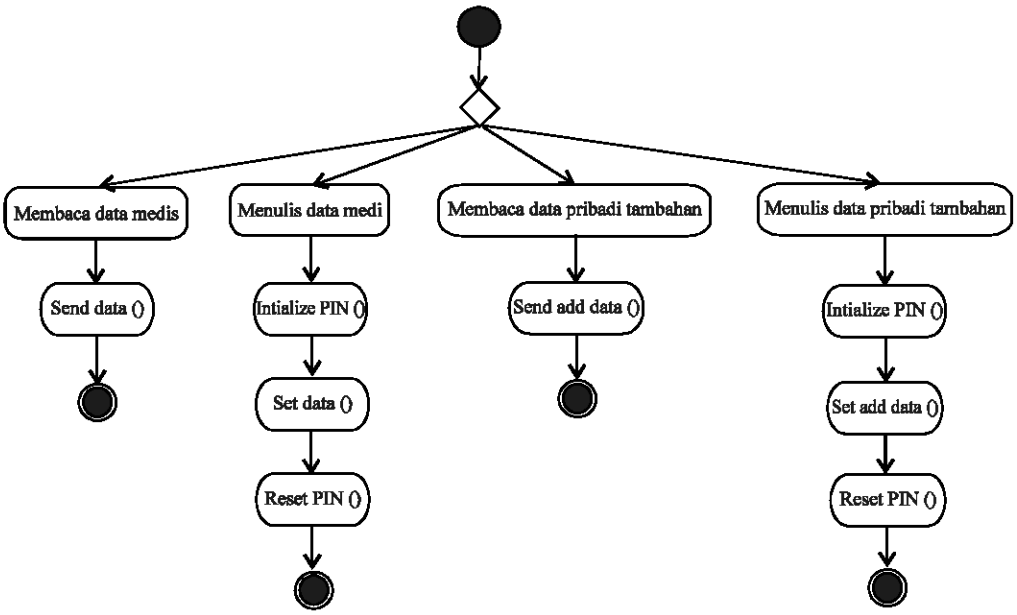

Fig. 6: JavaCard applet activity diagram

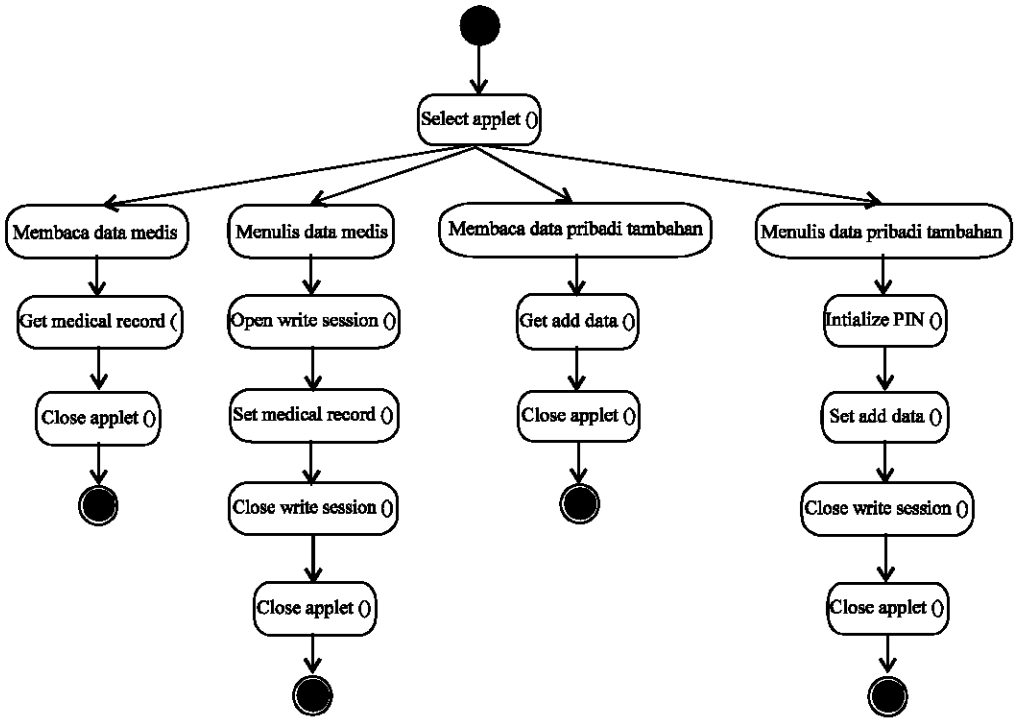

Fig. 7: Connector applet activity diagram

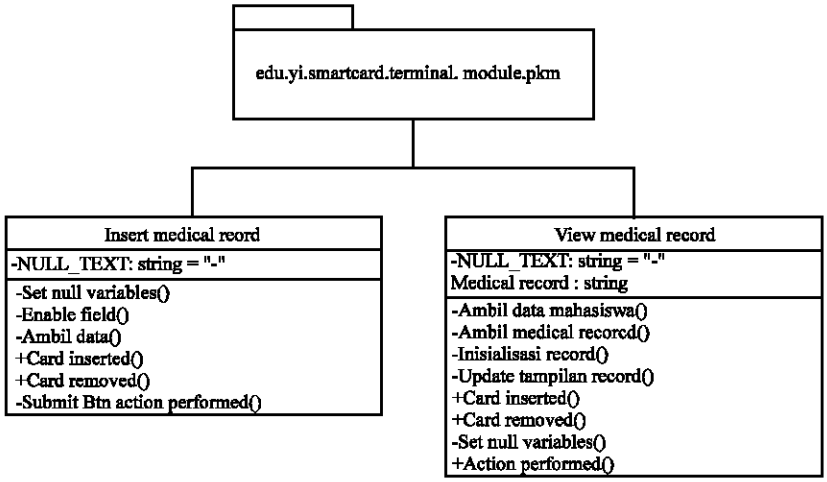

Fig. 8: Smart Health terminal module class diagram

In the connector applet, the operation begins with selecting the applet PKMApplet by checking its AID inputted by the user. If the user wants to read medical records, the connector applet will send a Command APDU to the card and the data will be taken from the JavaCard memory. If the user wants to write medical records, the PIN must be validated to begin the writing session. After the writing operation is finished, the session will be closed by resetting the PIN. Then, the applet PKMApplet will be closed. Reading and writing process for personal data use the same mechanism. Figure 7 shows the connector applet activity diagram.

Terminal module: Terminal module is an end user application to executes the Smart Health application. The terminal module is a Graphical User Interface (GUI) which consists of 2 parts, InsertMedicalRecord module and ViewMedicalRecord module. In the other word, Smart Health terminal module is a Java package contains of 2 class inside the package, InsertMedicalRecord class and ViewMedicalRecord class. Figure 8 shows the class diagram of the Smarth health Terminal module.

System integration: Smart Health system has three main components, the JavaCard applet, the connector applet

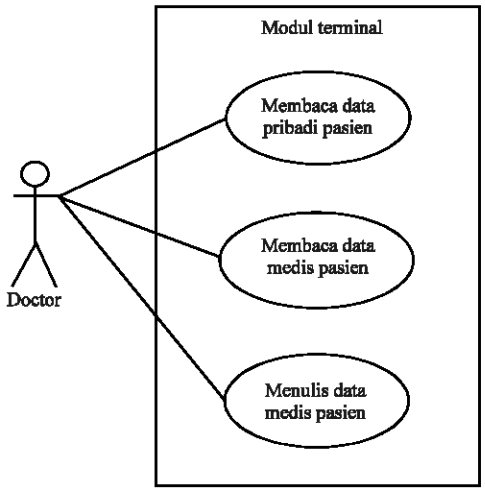

Fig. 9: Smart Health use case diagram

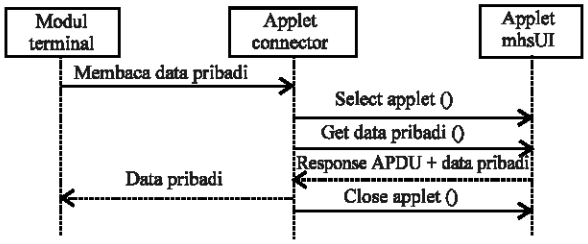

Fig. 10: Read personal data sequence diagram

and the terminal module. This components interacts one another and become the main requirement in making the Smart Health system. Without one of the component mentioned above, then the system will not work. The system integration is performed by wrapping the three main components into one Java Project.

The Smart Health application serves the user by providing the card holder's personal data, read his/hers medical records and write his/hers medical records. From the user side, Smart Health application is an easy to use application because the application only need few actions from the user. The user only need to input the JavaCard into the card reader and then the user can read or write medical records by accessing the field at the terminal module. Figure 9 shows the use case diagram of our Smart Health system.

Smart Health application contains 2 parts, reading the medical records and writing the medical record. To perform this 2 tasks, the applications must pass through three steps of activity. First, reading the personal data from applet MahasiswaUIApplet. Second, reading the additional personal data from the applet PKMApplet. Third, writing the medical record to the applet PKMApplet. Figure 10 shows the Sequence diagram or reading personal data.

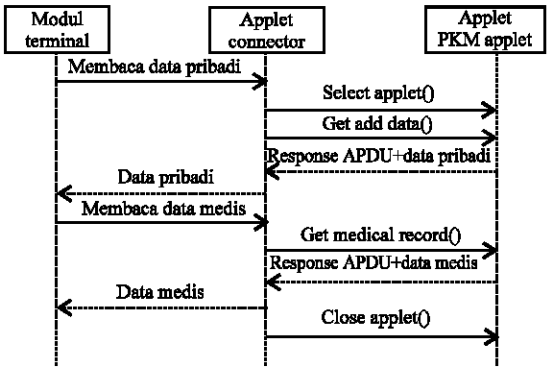

Fig. 11: Read medical record sequence diagram

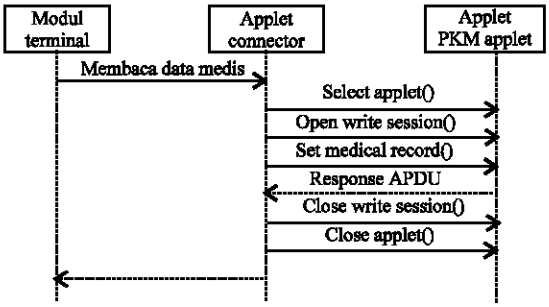

Fig. 12: Write medical record sequence diagram

The reading medical records process has some sequence to complete the operation. First, the terminal module will send a command to the connector applet to get the personal data from the JavaCard memory, inside the applet PKMApplet and the applet Mahasiswa UI Applet. Personal informations which is taken from the memory are the name, NPM, organization code, date of birth, sex and blood type. Figure 11 and 12 show the sequence diagrams of read and write medical record processes.

The next step is reading the citizenship and marital status information from the applet PKMApplet. After getting those data from the JavaCard memory, the last step is to read the medical records.

The medical records writing process also has some sequence to complete the operation. The first step is to get the personal data (name, NPM, organization code) which are stored in the JavaCard memory, inside the applet MahasiswaUIApplet. The next step, after the medical record is written in the terminal module field and submitted, the writing session will be started and then called the writing method. After the applet accepts a response from the JavaCard that the medical record has been stored, the writing session will be closed. The last step, the terminal module will tell the user through a dialog box.

SMART HEALTH IMPLEMENTATION

The keys of making a JavaCard application are the JavaCard applet programming and how to arrange the APDU to process and executes the operations of the application. The JavaCard applet is an integral part in making the application because the objectives are to store and read medical records from the JavaCard. To achieve the objectives, the need to arrange a good APDU format becomes important. The APDU format is used to control the reading and writing process so the operations will be done perfectly.

Smart Health could store up to 20 medical records and 2 additional personal data. Every medical record was divided into 6 parts, institution, Doctor ID, date, anamnesis, diagnosis and therapy.

The amount of memory needed as a storage media of the data above are 340 bytes for every medical record and 16 bytes for the additional personal data. The detail of the JavaCard memory allocation could be seen in Table 1. The total amount of memory needed to store all the data is 6816 bytes.

Below, the data flow of the Smart Health operation will be described. The writing process has two data flows, personal data flow from the JavaCard applet to the terminal module and medical data flow from the terminal module to the JavaCard applet.

In the personal data flow, the terminal module instructs the connector applet to access the JavaCard applet which then retrieves data from the JavaCard memory through APDU. The connector applet retrieves this data from the APDU then stores it in the dataPribadi() array. This array will be accessed and finally displayed in the terminal module.

In the medical data flow, data which is filled in the terminal module is stored by the connector applet in the data() array. Then, the connector applet instructs the JavaCard applet to write the medical data into the JavaCard. The data presented in the data() array is copied and filled in the APDU. Through the APDU, the medical data is then transferred to the JavaCard's memory, at the medicalRecord() array.

The medical data retrieval process has three data flows, i.e. personal data flow, additional data flow and the medical data flow from the JavaCard applet to the terminal module. In the personal data flow and the additional data flow, the terminal module instructs the connector applet to access the JavaCard applet, which then retrieves data from the JavaCard's memory through the APDU. The connector applet takes this data from the APDU and then stores it in the dataPribadi() and the dataTambahan() array. Both these arrays will be accessed by the terminal module and presented in the user interface. Figure 13 shows the data flow diagram of write medical record process.

In the medical data flow, the terminal module gives instructions to the connector applet to retrieve medical data stored in the JavaCard applet. The connector applet accesses the JavaCard applet, which then retrieves medical data from the JavaCard's memory through the APDU. The applet connector takes this data from the APDU then stores in the dataMedis() ar ray. The terminal module will then access this array and divide it based on available medical data categories, stored it in the medical Record() array and finally displayed it in the terminal module field. Figure 14 shows the data flow diagram of read medical record process.

Table 1: JavaCard memory allocation detail

Data category	JavaCard memory allocation (byte)
Institution	20
Doctor ID	12
Date	8
Anamnesis	100
Diagnosis	100
Therapy	100
Citizenship	15
Martial status	1

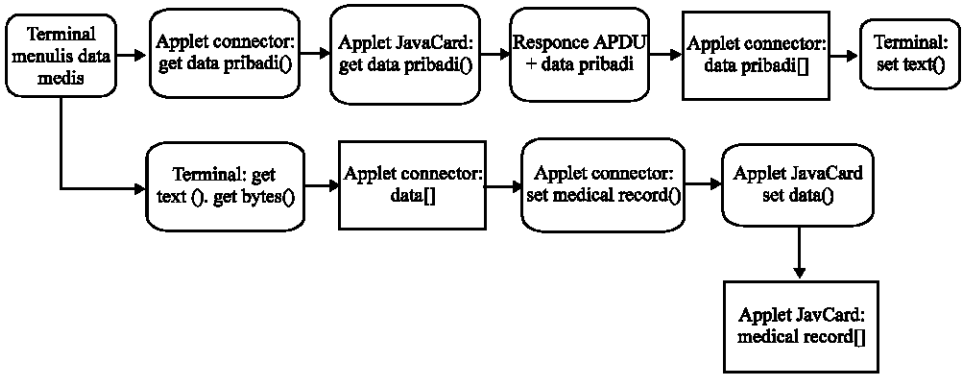

Fig. 13: Write medical record data flow diagram

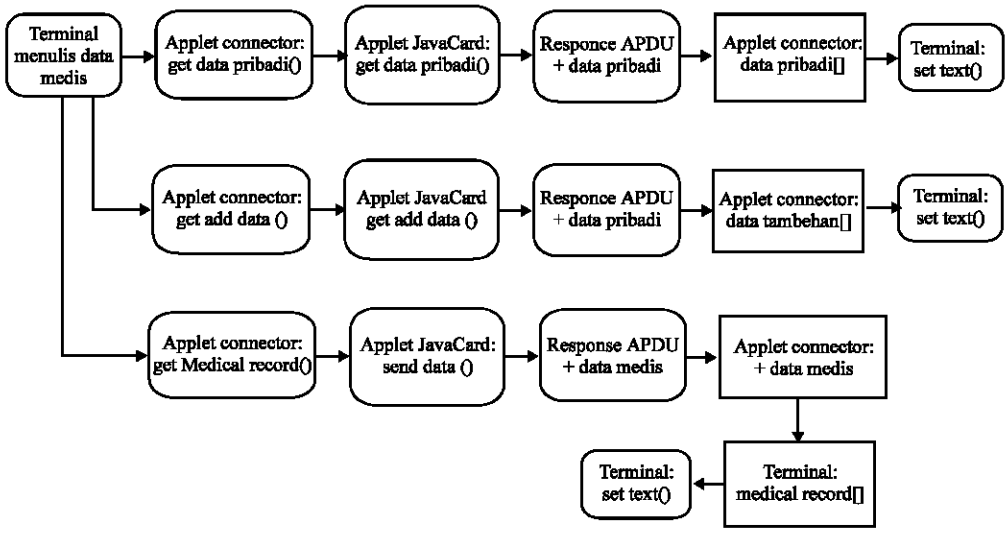

Fig. 14: Read medical record data flow diagram

SYSTEM PERFORMANCE EVALUATION

JavaCard memory allocation: The Smart Health application was developed using a Fix Length system where the maximum length of data which will be stored in the JavaCard memory is predefined and restricted to that maximum length (Table 2). Memory allocation for data storage on the JavaCard, depicted on Table 3, is considered adequate for representing a medical data record. Yet, some weaknesses arise from the usage of this Fix Length system.

The first weakness is when the length of data filled into the JavaCard is below its maximum. This condition causes void spaces in the JavaCard's memory which leads to memory inefficiency. Sections, which will likely caused memory inefficiency are the anamnesa, diagnose and therapy since the length of medical data for these sections can hardly be predicted. The second weakness is that when data filled into the JavaCard exceeds the maximum capacity determined. Incomplete data will be written to the JavaCard, causing missing information when data is retrieved from the JavaCard. Figure 15 shows the memory usage diagram.

The most relevant solution for tackling these weaknesses is by using Dynamic Length system in programming the application. Dynamic Length system insists that a program being developed must be able to detect the length of data being stored in the JavaCard. Data length is not restricted to a certain maximum length, so every bit of information can be stored. To separate data, special characters are used. On retrieval of data from the memory, the program will read the data strored in the JavaCard, until the special character is read. Thus, memory inefficiency and missing information can be avoided; optimal memory usage can be achieved.

Table 2: Fix length system efficiency

Memoty usage	Wasted menory (%)
20 medical records @ 280 byte	17.61
20 medical records @ 250 byte	26.41
20 medical records @ 220 byte	35.21

Table 3: Data comparison before and after compression

Data before compression	Data after compression	Percentage
6800 byte	4127 byte	60.7%
6000 byte	4136 byte	68.93%
5500 byte	4039 byte	73.44%

Data compression: The Smart Health application uses 6816 b of JavaCard memory to store twenty medical data records and additional personal data records. Every medical data has a size of 340 b and additional data 16 b which means 6800 b total is needed for all medical data and 16 b total for additional personal medical data. Memory calculation described above uses an assumption that every character inserted has a 1 b data length. Therefore, every medical data has a 340 character maximum length and a 16 character maximum for additional personal data.

Memory usage in the JavaCard is still open for further optimization by using data compression. With data compression, data size stored will be smaller since every character will need less than 1 byte of space. This leads to even better memory efficiency in the JavaCard. To implement the data compression, an additional library is needed as a tool for executing the data compression process. Compression is done at the connector applet,

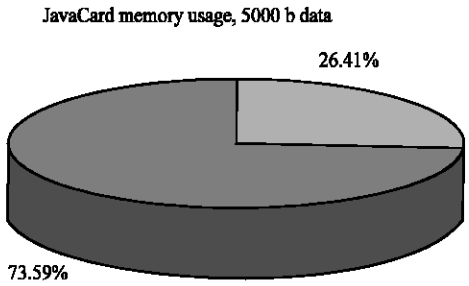

Fig. 15: Memory usage diagram using 5 kB data size

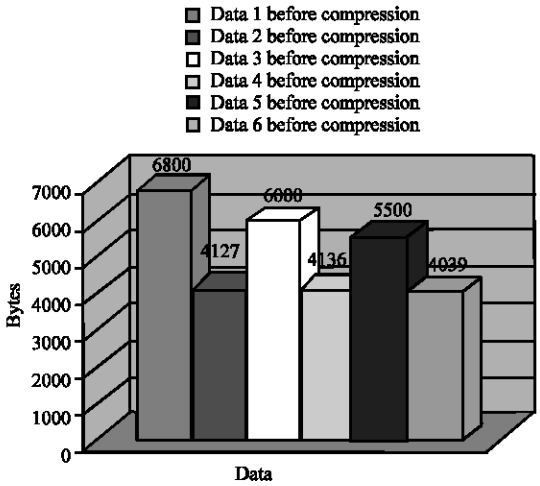

Fig. 16: Data comparison diagram before and after compression

which processes data array from the terminal module before stored in the JavaCard. In the data retrieval process, data taken from the JavaCard by the connector will be decompressed before being diplayed in terminal module. Figure 16 shows the comparison of the data before and after the compression.

JavaCard processing speed: The (erformance of the Smart Healh application is tightly related with the processing speed of the JavaCard. In medical data retrieval, the process done by the application is accessing the personal data and medical data stored in the JavaCard's memory. All data stored in this memory is read one by one, then displayed by the terminal module. In this process, the processing speed of the JavaCard will determine the overall performance of the application. The speed of JavaCard's microcontroller determines how fast the data retrieval process will be. As an example, data retrieval using JavaCard of the University of Indonesia's batch 2006 students needs about 20 sec. Using the JavaCard of the University of Indonesia's batch 2006 students, data retrieval time decreases to 12 sec. This means that the JavaCard UI 2007 performance is 40% better than the JavaCard UI 2006. Thus, the overall performance of the Smart Health application greatly depends on JavaCard's processing speed.

CONCLUSION

The making of a JavaCard application consists of three steps, JavaCard applet programming, connector applet programming and building the terminal module. JavaCard applet is an integral part in making the application and determines the continuity of the whole application operation. Connector applet is an applet that is used to connect the JavaCard applet with the terminal module. The main issue in making a JavaCard application are the JavaCard applet programming and how to arrange the APDU to process and execute the operation of the application. JavaCard memory allocation are also the main consideration to optimize the use of the JavaCard memory.

FUTURE WORK

Problems and limitation of this system has been described. Development of this system can be started from rechecking those problems so the performance of the application could be improved.

The Smart Health application can be improved in the future by applying the Dynamic Length system which can significantly improve the memory efficiency. Other than that, it is also expected that development of future applications implement data compression methods which even have greater impact on memory efficiency.

The scope of the Smart Health application could be broaden with the incorporation of a medical data record printing menu and with the integration of this application with database systems of certain health institutions through Web Service interfaces.

ACKNOWLEDGMENT

We thanks Adhi Yuniarto, Gladdi Guarddin, Jan Peter and Dimas, for their contribution and discussion on the wider implementation of UI Smart Card program.

REFERENCES

Chen, Z. and R.D. Giorgio, 1998. Understanding Java Card 2.0. Learn the inner workings of the Java Card architecture, API and runtime environment.

Chen, Z., 1999. How to write a Java Card applet: A developer's guide, Learn the programming concepts and major steps of creating Java Card applets. http://www.javaworld.com/javaworld/jw-07-1999/jw-07-javacard.html.

Chen, Z., 2000. Java Card™ Technology for Smart Cards: Architecture and Programmer's Guide. Addison Wesley.

Di Giorgio, R., 1997. Smart cards: A primer Develop on the Java platform of the future. www.javaworld.com/jw-12-1997/jw-12-javadev.html.

Di Giorgio, R., 1998. Smart cards and the OpenCard Framework, Learn how to implement a card terminal and use a standard API for interfacing to smart cards from your browser. http://www.javaworld.com/avaworld/jw-01-1998/jw-01-javadev.html.

Fodor, O. and V. Hassler, 2005. JavaCard and OpenCard Framework: A Tutorial. Information Systems Institute, Technical University of Vienna.

Guarddin, G. *et al*., 2007. http://smartcard.ui.edu..

Hartanto, A.A., 2007. Teknologi SmartCard dan Impian di Masa Depan. http://bebas.vlsm.Org/v11/refind1/physical/SmartCardDream.rtf,.

Surendran, D., 2000. Applications of Smart Cards. http://people.cs.uchicago.edu/~dinoj/smartcard/arch-1.html.

Surendran, D., 2000. Elements of Smart Card Architecture. http://people.cs.uchicago.edu/~dinoj/smartcard/Jcarch-1.html.

Surendran, D., 2000. Java Cards. ttp://people.cs.uchicago.edu/~dinoj/smartcard/JCarch1.html.