MDA Based Framework for the Development of Smart Card Based Application

Alireza Nikseresht, Koorush Ziarati

***Abstract*—Reducing time to market in spite of increasing the system's functionality and reuse of software on different platforms increase the need for revising the development methodologies for smart card based applications. The Model Driven Architecture (MDA) is a design methodology addressing these emerging requirements. This paper proposes a model driven framework for the development of smart card application. This framework allows developers to model the system in a high level modeling language without considering the target platform. The proposed framework transforms platform independent models to platform specific models which are closer to implementation and finally transform the models to executable source code, automatically. Furthermore, we describe the design and implementation of the proposed framework.**

***Index Terms*—** Smart Card, Smart Card Application, MDA, Software Development, UML, XML

I. INTRODUCTION

The closed architecture of today's smart card operation systems and lack of high-level Application Programming Interfaces (APIs), have turned card-application development to a very difficult and time-consuming task, requiring high specialized programmers and dedicated software tools [11]. Existing applications usually work only with a specific type of card, reader and platform, resulting in increased dependence on specific vendors and suppliers [10]. These emerging demands are addressed by the model driven architecture.

Model Driven Architecture (MDA) is an approach to software development that focuses on the production of high-level models that are used as the basis for automating system implementation [1]. As in all engineering disciplines there is a gap between system models and the actual system under construction. The goal of MDA is to reduce that gap by automating coding and implementation through the creation of knowledge-based tools. The MDA approach centers on the definition of a Platform Independent Models (PIM) using a high-level specification language. The goal is to develop models that are precise enough to support code generation, so that a PIM may be transformed into one or more Platform Specific Models (PSM) for the actual implementation. The advantage of the MDA approach is that models and code are more easily kept up to date and incremental, iterative development is facilitated by the direct transformation from model to code. While MDA is technically neutral about the syntax or structure of the high-level models, UML [16] has emerged as a common foundation for MDA modeling. UML has been used across a wide variety of domains, from computational to physical, making it suitable for specifying systems independently of whether the implementation is software or hardware. The recent addition of action semantics to UML has led to development of executable UML (xUML) which supports the direct execution of UML models [2]. In this paper we provide both a framework and a methodology that facilitates Smart Card application development, covering all Smart Card application specific requirements and needs. For this purpose the framework includes a set of tools and APIs to develop smart card application, thus ensuring cost-effectiveness, independence from suppliers and interoperability with existing systems and the web. The paper is organized as follows. Section 2 gives an overview on related works which separate target platform from Application development. In section 3 we describe the Architecture description. Section 4 covers the proposed system design methodology.

Manuscript received December 26, 2010. This work was supported in part by the Jahade Daneshgahi Fars.

A. Nikseresht is with the University of Applied Science and Technology Jahade Daneshgahi Shiraz, Iran (phone: +987112357294 e-mail: aniksereshtt@shirazu.ac.ir).

K. Ziarati is Assistant Professor of Computer Science and Engineering Department, Shiraz University, Iran (e-mail: ziarati@shirazu.ac.ir).

II. RELATED WORK

Three major APIs are available when you're writing client-side applications: the Open Card Framework [11], the Java Card RMI Client API [19], and the Security and Trust Services API (SATSA) [20]. The Open Card Framework (OCF) is an open standard, providing architecture and a set of APIs that enable application developers and service providers to build and deploy smart card aware solutions in any Open Card compliant environment [11]. The reference implementation has been developed using the Java programming language, so it is the ideal solution for application development in Java. For regulation of the smart card itself, Open Card relies on ISO-7816 [7], therefore it can be used together with any compliant smart card. OCF consists of the OCF Core, on which the several Card Service and Card Terminal components are plugged. The Card Services (supplied by the card providers) encapsulate the card operating system functionality and on card application dependencies, shielding the application from changes in those components. Accordingly, the Card Terminals (supplied by the reader providers) encapsulate the reader device functionality. This approach ensures that differences or changes in the card operating system, in the card terminal or in the application management scheme used by the card issuer do not impact the user application code [10, 11]. The applications developed against OCF high-level APIs can work with different cards and readers by just plugging the corresponding Card Service and Card Terminal implementations into the architecture. OCF also offers APIs

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

for card file management, including file creation, deletion, and update, as well as tools for establishing the necessary security mechanisms on the card, i.e. create Public Key structures, load/generate user keys and certificates, enable secure messaging functionality etc.

Java Card RMI (JCRMI) is based on the J2SE RMI distributed-object model. This approach provides an object-centric model, in which the APDU [7] communication and handling are abstracted. Instead, you deal with objects. This simplifies the programming and integration of Java Card technology-based devices. In the RMI model a server application creates and makes accessible remote objects, and a client application obtains remote references to the server's remote objects, and then invokes remote methods on them. In JCRMI, the Java Card applet is the server, and the host application is the client.

Security and Trust Services API (SATSA) is a set of optional packages for J2ME that defines a client-side API to access what are referred to as security elements: devices such as smart cards.

III. Architecture Description

The layered approach lets designers separate the application development from the platform and card implementation details. As shown in Fig. 1, the architecture comprises four separate layers, which establish the necessary interfaces between the system's distributed parts, from the Card to the end-user application.

In the following paragraphs we attempt a closer look to each layer, explaining the functionality and implementation details of the major structural modules.

A. Hardware Layer

Smart Card applications are not standalone, but rather part of an end-to-end application. A Smart Card application typically comprises four parts: 1-The back-end application that provides access to back-office services such as security or electronic-payment information stored in databases. 2-The host application which accesses Applications on the smart card using one of a number of interfaces for card access, such as the APDU commands or the Open Card Framework API, etc. 3- The card reader, card terminal, or card acceptance device, provides the physical interface between the host application and on-card Application. 4-The application part resident on the card henceforth called card-resident application.

The card-resident part comprises the data kept on the card in the context of an application. Thus, developing a card-side application involves building a file structure, where a single DF contains several DFs and EFs representing the required fields [4]. However, this approach has serious drawbacks: accessing these data requires a dedicated external application, which has knowledge of all card-side implementation details from the beginning. Furthermore, if a new standard evolves, the application file structure has to change completely in order to include new fields or remove others. This could be a daunting task, especially when talking for applications requiring large number of individual fields.

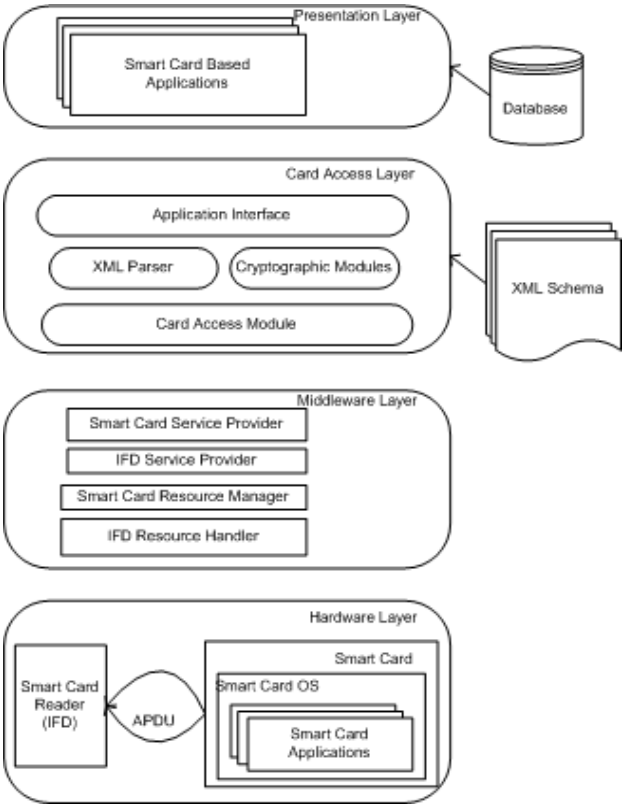

Fig. 1. The Overall Architecture of system.

Capitalizing on the latest developments of smart card technology (increased processing power and available memory); we use a document-based model for the development of card-side applications. Each separate application comprises an XML document, which includes all the necessary elements (fields) for its purpose of use. These documents are stored on single Elementary Files, one for each separate application. Thereby we avoid building complex card file structures, which is very difficult to modify or update in the future. The corresponding XML schemas for the document description, reside at the host site or on a public web location and they are handled in the upper layers of the architecture. Furthermore, external applications can access the card resident parts without having knowledge of the specific card-side implementation details. XML-based approach is a fast and convenient way of designing and deploying flexible and extensible card-side applications. Applications can be developed independently (by third parties) as XML documents. The cardholder can download them from the web directly to the card, through a procedure transparently handled by the Framework.

B. Middleware Layer

The primary goal of this layer is to hide complexity, while achieving maximum independence from card and reader manufacturers. This layer is based on PC/SC standard [18]. Currently, the use of Smart Cards in the PC environment is hampered by the lack of interoperability at several levels. First, the industry lacks standards for interfacing PC to IFD (Interface Device or Card Reader). This has made it difficult to create applications that can work with IFD from a variety of vendors. Attempts to solve this problem in the application domain invariably increase costs for both development and maintenance. It also creates a significant problem for the PC user in that an IFD used with one application may not work with future applications.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Second, there is no widely accepted high-level programming interface for common Smart Card functionality [18]. Encapsulation of Smart Card interfaces can dramatically simplify application development and reduce costs by allowing low-level interface software to be shared across multiple applications. In addition, a standardized high-level interface allows applications to reduce their dependency on a specific Smart Card implementation, making it far more likely that an application will be able to use future, enhanced Smart Cards. To increase the flexibility and independence of PC-based software components necessary to deal with multi-application Smart Card, it is also important to reflect the different roles of Smart Card vendors and Smart Card issuers through the mechanisms provided for Smart Card application deployment. Only by separating Smart Card technologies from the Smart Card applications can specific Smart Card technologies become transparent to Smart Card based PC applications when dealing with Smart Card applications. Third, mechanisms to allow multiple applications to effectively share the resources of a single Smart Card are not defined. These are critically important as we rapidly move toward the deployment of multiple-application Smart Cards and generic cryptographic Smart Cards that will be used as part of a multiprocessing PC environment. Without agreed upon standards for device sharing, it becomes effectively impossible for application developers to ensure that they can complete an operation using Smart Card services without interruption.

To optimize the benefit to both the industry and end users, it is important that solutions to these issues be developed in a manner that supports a variety of operating environments and a broad base of applications. Only through this approach can we support the needs of all constituencies and encourage development of Smart Card-based PC applications, as a cost-effective solution to meeting requirements in a very diverse set of markets.

Smart Card technology offers a vital addition to the security infrastructure of the PC and network environments. It is an enabling technology for network commerce in general. To achieve this potential, however, it is essential that a consistent framework exists into which the diverse efforts of application developers, network technology vendors, and Smart Card technology vendors can be coherently channeled. PC/SC is a standard addressing these emerging requirements. Because this layer is based on PC/SC, it inherits all these property from PC/SC.

C. Card Access Layer

The Card Access Layer sits between the Middleware and the end-user applications, handling the interaction with the card-residing XML applications. In fact, it turns row data communication to XML documents exchange. It includes three basic modules: 1-The Card Access Module provides the tools for accessing the card-residing XML applications. The Card Access Module is responsible for retrieving the stored information in the form of an XML document. 2- The XML Processing Module provides the necessary functionality (e.g. XML parser) for processing the XML documents. It also includes XML compression tools, which can be used to reduce the documents Size in order to overcome card memory limitations. 3- The Cryptographic Module provides additional, application-level security features. 4- The Application Interface provides high level functions to access on card data, this is an interface between front-office application and other layer that allows front-office application have a transparent view of the smart card.

D. Presentation Layer

The upper layer of the architecture is the Presentation Layer. It includes the various front-office applications that make use of Smart Card.
Based on the Card Access Layer functionality, the user applications interact with the smart card transparently. Developers can independently deploy Graphical User Interfaces (GUIs) for interacting with the card applications, using various technologies and tools.

IV. System Design

Software modeling is becoming a critical process in software development. Modeling technologies have matured to the point where it can offer significant leverage in all aspects of software development [15].
For example, the Unified Modeling Language (UML) provides a rich set of modeling notations and semantics, and allows developers to understand, specify and communicate their application designs at a higher level of abstraction [16].

The notion of model-driven development aims to build application design models and transform them into running applications [13]. Given modern modeling technologies, the focus of software development has been shifting away from implementation technology domains toward the concepts and semantics. System design in our approach as shown in Fig. 2 is guided by the idea of the MDA introduced by the Object Management Group (OMG). The primary goals of the MDA are portability, interoperability, and reusability of applications achieved by architectural separation of concerns [3]. For separating the application logic from the underlying platform, models with various levels of detail and focus are used. A model is a formal specification of a system and provides an abstraction, i.e. the model includes certain classes of information while suppressing other ones. The selection of which classes of information to include or

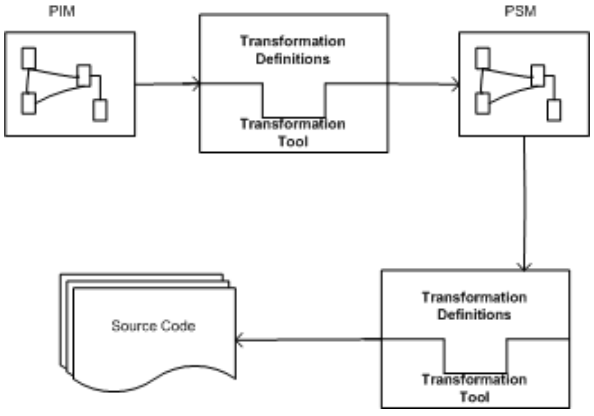

Fig. 2. System Design Proces

suppress depends on the purpose and the focus of the model. A particular selection of such information classes is denoted as a viewpoint. Widely used viewpoints in the design of distributed computer systems are platform independent and platform specific viewpoints, which separate the application logic from the underlying platform technology. The MDA

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

proposes such viewpoints – denoted as Platform Independent Model (PIM) and Platform Specific Model (PSM) – and defines their role in the design of a system [3].

A. Platform Independent Model

A PIM is a formal specification of the structure and function of a system that abstracts away technical details [16]. In our approach the PIM structures the overall application into data to be stored on card and methods that work on data. So the PIM contains some classes and their relations, which expressed in unified modeling language.

B. Transform PIM to PSM

A transformation between two models roughly consists in sets of rules that establish correspondences between elements of the source model and elements of the target model [8]. The MDA approach is used to generate internal representations by means of model-based transformations, implemented using QVT (Query, View, Transformation) resources, standardized by OMG [14]. Figure 3 presents the general scheme for model-based transformation, and shows that transformations could be themselves considered as models. Then, the concepts to be used in order to define transformations must be defined using a meta-model.
The transformation of the PIM to PSM is accomplished by defining some specifications of the target platform. Specifications that are defined in this step are: 1- Card type. 2- Access privileges for each data elements or classes. 3- Atomic operations [6]. After defining these specifications we can start the transformation to produce PSM.

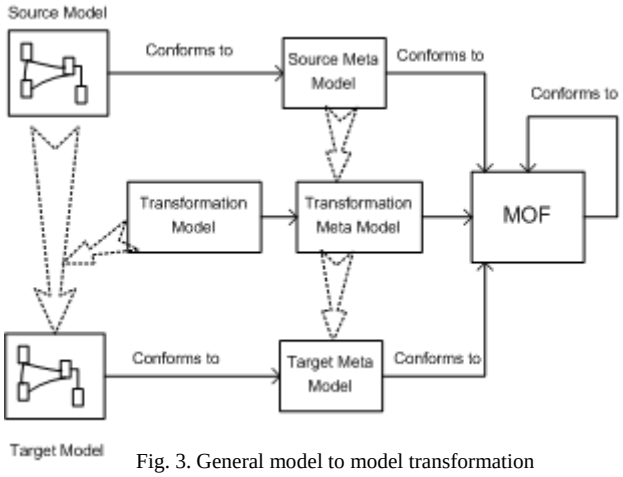

Fig. 3. General model to model transformation

C. Platform specific Model

A platform specific model extends the specification in the platform independent model with the details that define how the system operates with the target platform. The PSM extends PIM with the following information: 1-Some useful methods to read/write from/to card, based on specification that are defined in transformation. 2- Allocation of data elements to on card XML files. 3- Cryptographic methods.
4- Additional constrains. Object Constraint Language (OCL) provides means for formally specifying additional constraints [12], without enhancing the complexity and reducing the readability of the UML models. OCL is a formal language for describing expressions on UML models. It can be used to specify invariants that must hold during the whole lifetime or only in particular states. With OCL, constraints at model level and meta-model level can be described.

V. Code Generation

Code generation is the transformation of the PSM to code considering the target platform. Our code generation strategy is based on templates.

The code generator uses different templates according to the specification in the PSM. Code generation starts with defining the target programming language and defining the middleware as we saw in section 3. The code generator imports PSM models as XMI [17] descriptions and parses them to constructs their in-memory objects, and then validates [5] the models. After that it generates an intermediate objects based on the transformation definitions. Finally the code generator generates final compatible source code and XML schemas based on one of the templates.

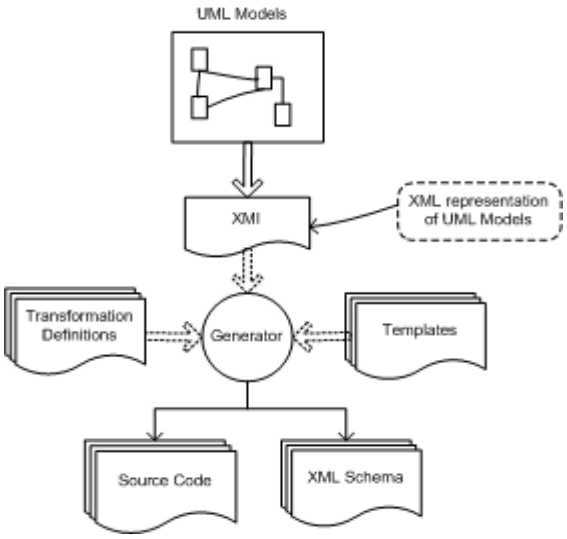

Fig. 4. Code Generation.

VI. Case Study

Patient Health Card System mission is to provide emergency medical information for each patient which is stored on a smart card and carried by patients. One of the key points to implementing this system is considering standards in different layers. In this project we used PC/SC standard for communication between application and card reader. For communicating with card we used ISO7816 standard.

This Case Study is implemented in Shiraz Kosar hospital. In this project we used our framework to design and generate required source code. The generated code used without any changed and it worked fine. This test showed that it is a good framework to doing this kind of job. In this system all patient emergency information will be saved on smart card. Each patient has its own smart card that contains his/her information. A copy of card information will be saved on Hospital central database. This information consists of patient personal information, Emergency Contacts, Drugs which is used by patient, the patient doctors, patient hypersensitive and patient disease history.
The smart card that we used in this project was an 8 KB crypto memory smart card.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The goal is implementing some methods to store and retrieve the patient information on smart card in a certain security condition.

According to our presented approach, first we should design the PIM. The PIM consist of class diagrams and their relations. We can see the patient Health Card System PIM in figure 5. To draw the PIM models we use Argo UML which is a free and open source tool. Argo UML can import/export XMI files. This ability in addition to being open source caused that we choose this tool. After designing PIM, we must transform PIM to PSM. In this step the transformation tool input PIM in XMI format The PSM file that is generated can be edit by any tool that supports XMI. The XMI file can be used in next step without change. However, if you want you can change it and applying corresponding transformation rules to generate PSM.

The Generated PSM is in XMI format, and it can be imported by any tools that have ability to import XMI files. We imported PSM XMI file in Argo UML to show the PSM class diagrams. In figure 6 you can see the PSM class diagrams.

The next step is code generation. In this step we import PSM XMI file and generate the target source code. Then you can easily include this source code in your program and use it. The generated source code programming language is compatible with Borland C++ Builder.

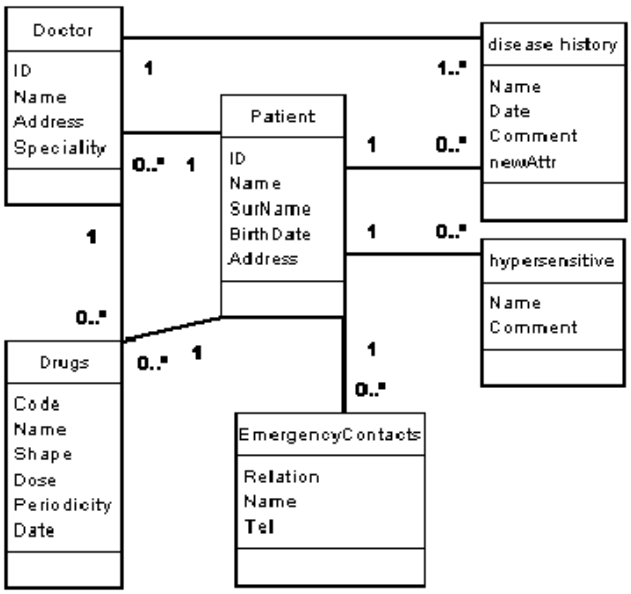

Fig. 5. PIM class diagram.

In transforming PIM to PSM we will create an application class. This class is in charge of communicating with card and reader. Application class hold information about application and application file structures. Application class

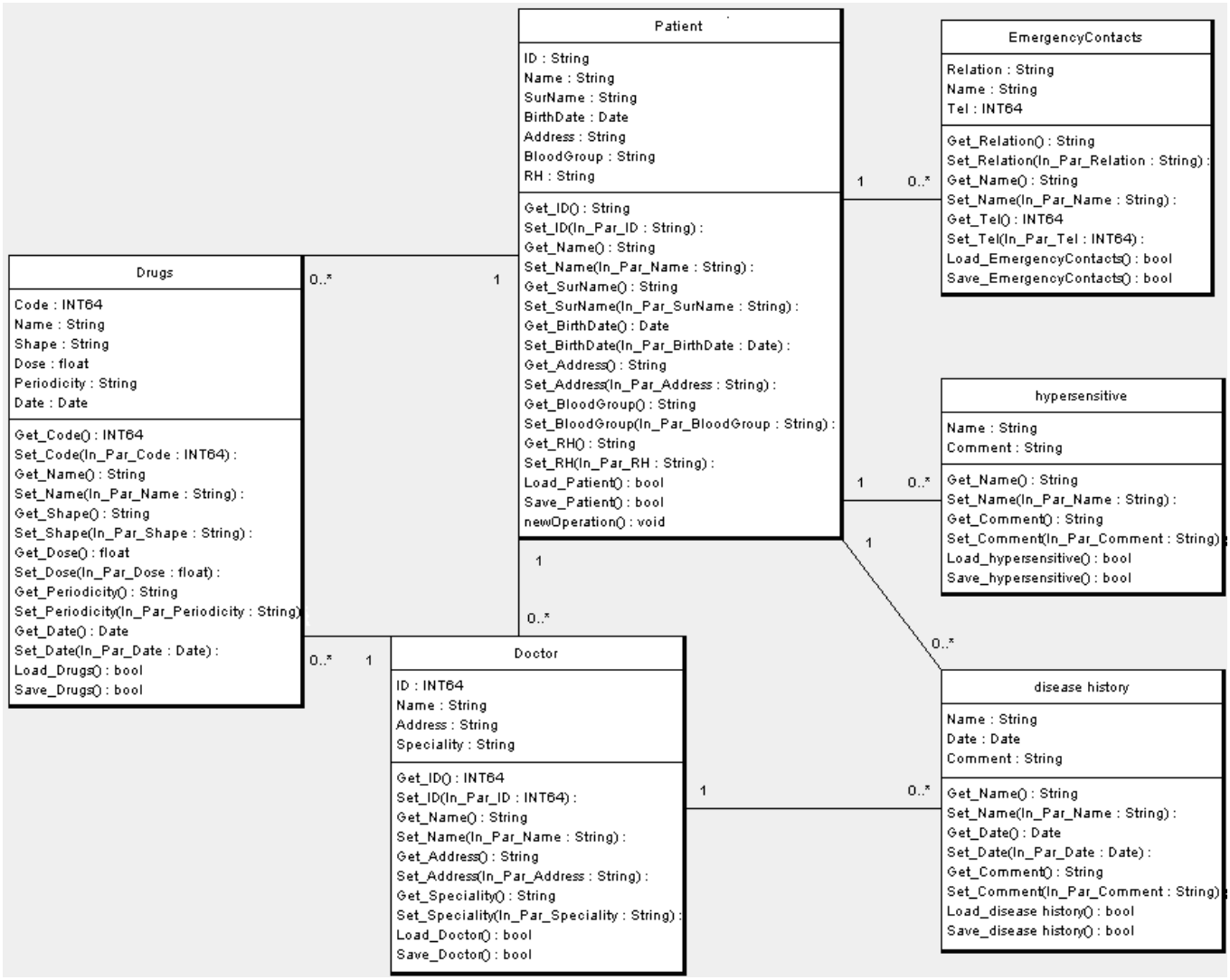

Fig. 6. Generated PSM class diagrams.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

is related to an application XML file which is stored on card. This file contains general application information and file structures. You can see a simple copy of this file in figure 7.

```
<Application>
  <Provider>Shiraz University</Provider>
  <Version>1.0</Version>
  <ID>100101564</ID>
  <Name></Name>
 <EFs>
    <Patient  FileType="Transparent" FileID="0XE1"/>
    <Doctors  FileType="FixRecord" FileID="0XE2"/>
    <Drugs  FileType="FixRecord" FileID="0XE3"/>
    <Hypersensity  FileType="FixRecord" FileID="0XE4"/>
    <Desiase  FileType="FixRecord" FileID="0XE5"/>
</EFs>
</Application>
```

Fig. 7. Application XML file.

For each application we will create a DF which is contains some EFs. For each class we will create an EF. If the class multiplicity is greater than 1 then we chooses record oriented file structure. But if the class multiplicity is 1 we choose transparent file structure. In figure 8 you can see the patient health card file structure which is located on smart card.

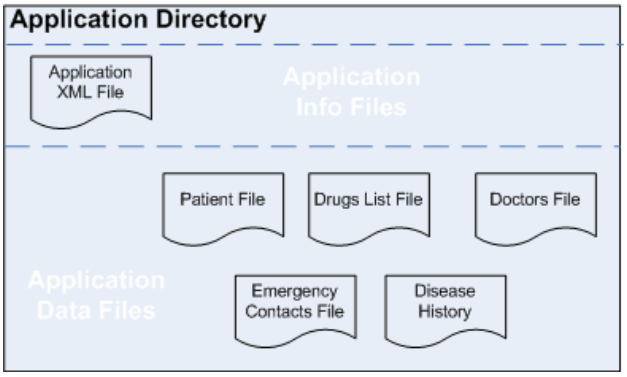

Fig. 8. Application file structure.

VII. Conclusion

This paper presents a framework for developing cost-effective, flexible and fast smart card based application that make use of ISO 7816- compliant smart cards, on different target platforms. This framework allows developers to model the system in a high level modeling language and to transform them toward final source code in a model driven manner. Without this framework you would have to know the specification of both the card and the card reader, further, you would have to write code which communicates only with this specific type of terminal and card. The generated source code is very reliable and out of errors so it save a lots of time in testing the system. Using this framework will enhance the applications and services available to the end consumer, enhance the marketplace for application developers, enhance the marketplace for network technology vendors, and enhance the marketplace for Smart Card technology vendors.

Acknowledgment

Authors gratefully acknowledge the D.G.Dena Company which has provided the facilities to implement the case study in Shiraz Kosar Heart Hospital.

References

[1] Anneke Kleppe, Jos Warmer, Wim Bast. "MDA Explained : The Model Driven Architecture: Practice and Promise." Addison Wesley, April 21, 2003.

[2] Stephen J. Mellor, Marc J.Balcer. Executable UML A Foundation for Model Driven Architecture. Addison Wesley, May 2002.

[3] OMG, "MDA Guide Version 1.0.1", Jun 12, 2003. http://www.omg.org/mda.

[4] Wolfgang Rankl and Wolfgang Effing."Smart Card Handbook Third Edition",John Wiley & Sons, 2004.

[5] Bhuvan Unhelkar, "Verification And Validation For Quality of UML2.0 Models", Wily, 2005

[6] Silberschatz, Korth and Sudarshan , "Database System Concepts fifth edition" 2006.

[7] ISO/IEC 7816-4, "Identification Cards, Integrated Circuit(s) Card With Contacts",Part 4,1995.

[8] Stephane Bonnet1, Olivier Potonniee1, RaphaÄel Marvie, and Jean-Marc Geib "A Model-Driven Approach for Smart Card Configuration ", Lecture Notes in Computer Science Publisher Springer Berlin / Heidelberg , ISSN 0302-9743, Subject Computer Science, Volume 3286/2004,Book Generative Programming and Component Engineering, 2004.

[9] A.Georgoulas, A.Giakoumaki, D.Koutsouris "A Multi-layered Architecture for the Development of Smart Card-based Healthcare Application", Engineering in Medicine and Biology Society, 2003. Proceedings of the 25th Annual International Conference of the IEEE, Sept. 2003

[10] IBM "OCF 1.2 Programmers Guide", Forth Edition December 1999.

[11] U.Hansmann, M.S.Nicklous, T.Schack,F.Seliger ,"Smart Card Application Development Using Java",Spring 2000.

[12] J. Warmer and A. Kleppe "The Object Constraint Language Secound Edition: Getting Your Models Ready For MDA." Addision-Wesley,2003.

[13] S.Sendall and W.Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development" IEEE Software, vol 20 no 5, Sep./Oct. 2003

[14] MOF QVT - QueryViews/Transformations. OMG doc. ptc/2005-11-01, Nov. 2005.

[15] B. Selic, “The Pragmatics of Model-Driven Development” In IEEE Software, vol. 20, no. 5, September/October, 2003.

[16] Object Management Group, UML 2.0 Superstructure Specification, http://www.omg.org/, 2004.

[17] Object Management Group, MOF 2.0 XML Metadata Interchange, http://www.omg.org/, 2004.

[18] PC/SC Standard, " Interoperability Specification for ICCs and Personal Computer Systems",Revision 2.01.01 September 2005.

[19] Java Card Documentation http://java.sun.com/products/javacard/RMI_Client_API.pdf

[20] Enrique Ortiz. The Security and Trust Services API (SATSA) for J2ME: The Security APIs. http://developers.sun.com/techtopics/mobility/apis/articles/satsa2. September 2005.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)