Java Smart Cards as a Platform for Electronic Commerce

Joachim Posegga[*]
Deutsche Telekom AG
Technologiezentrum
IT Sicherheit/FE34a
D-64276 Darmstadt
Posegga@tzd.telekom.de

Abstract

This paper gives a brief introduction into Java for smart card; it outlines the basic concepts, the architecture of Java cards and the surrounding system, and briefly sketches security issues.

It is argued that Java smart cards form a novel platform for applications in the customer's wallet, that is well suited for applications in the context of Electronic Commerce.

1 Introduction

One of the advantages of writing a paper about platforms is that it can largely be avoided to define the exact nature of applications to run on these platforms. Thus, the author can elegantly escape the problem of defining what "Electronic Commerce" precisely means, and leave this task to others.

We will consider a technology that is closely related to Electronic Commerce, namely smart cards. Quite unnoticed from most of the computer science community, smart cards evolved in the last few years from comparably simple devices with hard-wired functions (eg: authentication) towards fully-fledged computers: Some high-end cards appearing on the market in 1998 feature a 32-Bit RISC processor, 32 K of memory, and –according to their manufacturer– 20 MFLOPS of computing power. This is comparable to a processor of a high-end PC just a few years ago. Manufacturers even think of producing cards with input and output devices, like a keypad and an LCD-display, in the future.

The software side of smart cards is evolving in parallel: For a long time, programming of applications for smart cards was only possible for highly specialised experts. All applications had to be written in machine language, the executable code even went into the card's ROM, forcing developers to tightly cooperate with chip manufacturers. This resulted in extremely high barriers to enter the market of smart card applications.

[*] The opinions expressed in this paper are those of the author and do not necessarily represent opinions of Deutsche Telekom AG.

Currently, smart card software resembles a development that took place in computer science some twenty years ago: when computing power became cheaper and resources more plentiful, the market allowed to trade efficiency of software for ease of its development: higher-level programming languages became established, and writing applications for computers evolved from alchemy to wide-spread routine. In the case of smart cards, the same is happening right now: in 1998 we will see the first practically usable cards that support Java, i.e.: provide a platform to run Java applets inside the smart card.

This development can be seen as a paradigm shift, and it will almost certainly change the smart card business significantly: it will become much easier to develop smart card applications, so everyone with some basic knowledge of Java and some understanding of smart card particulars (like APDU-processing) can develop a smart card application within a few days or weeks. The "traditional" players of the field will be confronted with new competitors as well as with new application areas. In parallel, the market for smart card applications can be expected to increase significantly.

Thus, Java smart cards could in fact revolutionise the smart card business and open smart card technology to wide-spread use.

2 Java for Electronic Commerce

We will first briefly consider what Java in general promises for Electronic Commerce, before going into the details of Java smart cards.

No matter what concrete Electronic Commerce application one is interested in, there are two quite obvious advantages a Java-based implementation can offer: firstly, the flexibility of downloading the application to the customer's computing platform on demand, and second the advantage of a platform-independent solution.

The principle of downloading code on demand avoids a serious problem of "traditional" client/server scenarios, namely the bottleneck for innovations that "classical" clients suffer from: Changing these (resp. their software) in order to match new or updated services can turn out to be arbitrarily difficult and expensive. Java solves this problem elegantly, since the clients' software is not tightly coupled to the client any more.

The advantages of platform-independent solutions are straightforward, since the effort required to maintain and run a service in a heterogeneous network is drastically reduced.

Thus, Java-based implementations have clear advantages; the question, *What is a suitable Java platform for these applications?* remains.

2.1 Java Smart Cards as Application Platforms

Java Smart cards form, literally, the thinnest Java client one can think of; if a device has a smart card interface, then a Java card is probably the easiest and cheapest way to add a Java support to it: one can expect costs between 5 and 20 US-$ for a Java smart card when mass production will be starting.

Using Java smart cards in some device requires, of course, that the device provides suitable interfaces to the card, and that the card implements a corresponding API. Both prerequisites are not yet fulfilled by the time of writing this, but the technology is about to be established.

There is still another obstacle against smart cards, namely sparsely found smart card readers. However, also these will become more widespread in the future, since prices of card readers decreased significantly. Mobile phones and many public phones already come with an integrated smart card reader, and there is a clear trend that smart card readers will become an essential part of many telecommunication terminals and also a standard for future PCs.

One can therefore expect a significant growth of smart card technology in the future. As soon as one major card issuer (e.g. a credit card company, or a network operator) switches to Java-based cards, plenty of Java platforms waiting to run applications to run on them will be found in the wallets of prospective customers.

But what makes Java smart cards so attractive as application platforms, especially compared with direct Java support in personal appliances? Essentially, there are three reasons:

- Java smart cards are cheaper than directly embedding Java in devices, therefore the technology can spread faster.
- Smart cards can be used in different environments (phones, computers, settop boxes, etc.).
- A smart card is personalised to its owner and it can be regarded as a comparably tamperproof device, suitable for processing “sensitive” applications, e.g. those involving flow of money. Furthermore, it also offers a secure environment to applications, in the sense that these cannot by analysed or re-engineered (if applications are loaded as cyphertext into the card—see section 3.2).

For these reasons, Java smart cards form an attractive platform for Electronic Commerce applications. They are of course not a panacea, and one should also be aware of their limitations: Smart cards offer little resources and will probably be never as powerful as processors in appliances “around” smart cards. Therefore, smart cards are not suitable to run computation-intensive applications. However, there are many application areas where comparably little processing power is required, and for these Java smart cards can form a well-suited platform.

The rest of the paper will now turn to some more technical issues and shall explain the architecture of Java cards and their use. The primary goal is to provide an introduction into the technology, in order to allow the reader to judge the appropriateness of Java smart cards for possible applications, and to give a starting point for further investigation of the subject.

3 The Architecture of Java Smart Cards

At the time of writing this paper, we see the first prototypes of Java smart cards appearing on the market. These are based on 16K or 32K smart cards, with an 8 or 16 K EEPROM and 256-512 Bytes of RAM.

Figure 1 outlines the architecture of a smart card supporting Java: As usual, a smart card processor forms the core of the card, running some smart card operating system. The Java virtual machine is implemented on top of the operating system: essentially, it consists of the Java interpreter and the Java Card API. This API allows applications in the card to access to the smart card internals (e.g. an ISO 7816-4 file system), and to the outside world (e.g. APDU-handling). There might also be domain-specific APIs, like for financial or GSM applications, or crypto-functions, which are not necessarily part of a Java smart card as defined by the Java Card Forum (http://www.javacardforum.org).

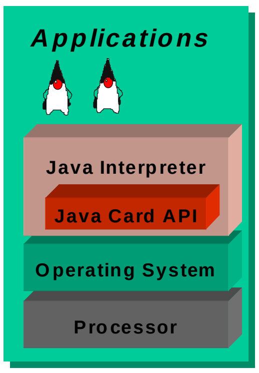

Figure 1: ***Java Card Architecture***

The card’s hardware consists of a processor with several types of memory: ROM, EEPROM, and RAM. The operating system and the Java VM usually lives in the ROM and partly in the EEPROM; Java applets are stored completely in the EEPROM, since they have to be exchangeable.

3.1 Programming with Java for Smart Cards

There are some differences between programming Java smart cards and “normal” Java applications: several features of Java are not usable inside a smart card, like multi-threading or dynamic class loading. Large data types like floats or double integers are also usually missing. Furthermore, one also has to be quite careful with allocating run time storage, since there is no garbage collector. Last but not least, the characteristics of smart card I/O need to be considered when programming applications.

Overall, programming an application for a Java smart card is related to the usual programming in Java, but there are significant differences and particulars to consider. A good Java programmer should not take more than a few weeks to become acquainted with the new application area. Delivering a secure application, however, is another issue (see Section 4 below).

3.2 Loading Applications into Java Cards

The careful reader will have realised that we did not mention security-related components of Java like the byte code verifier [10] so far. It’s purpose within the Java security architecture is to ensure that applications conform to certain rules, like not producing type violations or overflowing the operand stack of the Java VM at run time. The Java interpreter running applications does not check these rules any more, chiefly for the sake of efficiency.

Smart cards, however do not leave enough room inside the card for a byte code verifier inside the card. Two possible solutions for this problem exist: Performing the tests during run time, or establishing a byte code verifier outside the card. The first Java card that appeared on the market [3] chose the first option, but it seems like technology is moving more towards the second solution.

Figure 2 outlines the architecture for loading applications into cards that seems to turn into a standard for Java cards: First, the application to be loaded has to pass a bytecode verification step. The byte code is then converted into a so called “CAP”-File, which is, essentially, a more compact but semantically equivalent representation of the byte code. This CAP-File is finally loaded into the card, usually by secure messaging or in a digitally signed format to ensure that the application comes from a trusted source.

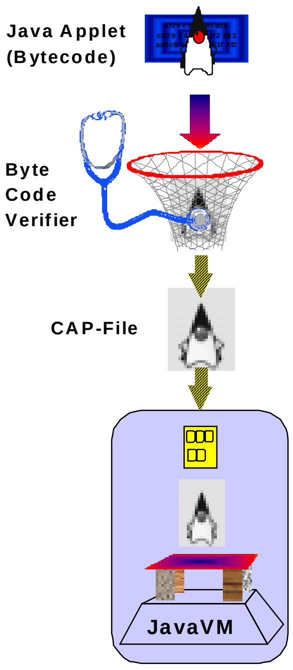

Figure 2: ***Loading Java Card Applications***

4 Security of Java Smart Cards

Security is certainly a very important issue for smart card applications: cards usually contain secret keys and it is the card issuers’ nightmare that these keys “leak” to the outside of the card. What influence does Java have to the security of smart cards? There are several arguments to consider:

The “traditionalists” argue that Java makes the situation worse, since it complicates the overall scenario, which in turn can possibly lead to more opportunities for attacks. Furthermore, it is argued that it is harder to secure a smart card where the software running inside the card is exchangeable.

The Java enthusiasts, on the other hand, give counter arguments: They claim that the additional “layer” of the Java virtual machine inside the card means more security, since it forms a strong barrier between applications and sensitive data like secret keys. Furthermore, the option of loading applications into the card comes handy for fixing bugs or security holes in applications, even *after* cards have been delivered to customers.

All these arguments for and against Java for smart cards are to be taken seriously, but it is currently not possible to come to a final conclusion on how to weigh them against each other. One has to wait until more experience with the technology has been gained.

Clearly, the correct implementation of the Java VM inside the card, and a correctly implemented system outside the card for distributing and loading applications are crucial to security. These issues have to be taken more seriously that in other application areas like Web browsers, where several security-related bugs have been seen in the past [4,8]; bugs in a Java VM inside a smart card would be much more serious, since applying patches to the Java VM in smart cards that are already delivered to customers required to physically exchange the smart cards. This is an extremely complicated and expensive undertaking.

One way to tackle this problem could be formal methods, which can help to avoid malfunctioning of software. Sceptics of formal methods often argue that these techniques are hard to apply in practise and that they do not scale to real-world problems. Software for smart cards, however, seems well suited for a formal treatment: applications are small and do hardly carry out complex algorithms. Moreover, several programming constructs (like threads), that are hard to treat formally, are missing in Java for smart cards. Thus, the setting of Java for smart cards seems quite ideal for the application of formal methods. Some attempts in this direction have are already been made [1,2,7], and it will be seen to what extend formal methods can contribute to maintain a high security standard for Java smart cards.

Even the most secure system design for Java smart cards cannot, however, prevent flaws in the design and implementation of the actual application running inside the card. Java for smart cards makes implementing applications easier, so we will certainly see new developers with little experience coming up. Here lies probably the highest risk for security holes: Although implementing a Java smart card application becomes easy, designing it securely will remain to be a task that requires extensive experience and in-depth know-how. Using Java does not help at all in this respect.

5 Conclusion

The paper gave a brief introduction into Java for smart cards, outlining the basic concepts, the architecture of the cards, the surrounding system, and briefly sketched security issues.

Java smart cards can be expected to change the smart card business significantly: it becomes much easier to develop applications, which will significantly increase the smart card (applications) market. New application developers, and also new applications will be seen. Two things, however, will not become easier: The design of a secure application, and the logistics behind smart cards (personalisation of cards, key management, TTPs, physical delivery of cards, etc.). Especially the latter will remain a sophisticated task requiring extensive experience and an expensive infrastructure.

The interesting perspective of Java smart cards for Electronic Commerce is that the technology can be seen as a new application platform in the customer's wallet, suitable for running applications in a comparably secure environment. This platform can, in principle, be used in different environments, like phones, computers, settop boxes, etc. The corresponding interfaces to telecommunication terminals and computers is currently being standardised, and the technology will penetrate products in foreseeable time, probably in the GSM-sector, first.

The technology of Java smart cards not yet fully developed, but it could support Electronic Commerce applications and help turning them into a success on the market. This requires that the major card issuers understand the opportunity that is offered: Java smart cards can form a platform for applications that is comparably easy and convenient to use, but the logistics for bringing these platforms adequately to customers will remain a high barrier; opening existing platforms to third parties could turn out to become a significant source of revenue in the future.

There is, however, one major non-technical problem yet to be solved: The widespread use of smart card technology requires a suitable, world-wide cryptographic infrastructure. This will hardly be possible without compatible legal regulations; resolving this is a task politicians should devote much more attention to.

References

1. E. Börger and W. Schulte: Programmer Friendly Modular Definition of the Semantics of Java, in: J. Alves-Foss: Formal Syntax and Semantics of Java, Springer LNCS, 1998.
2. R. M. Cohen: *The Defensive Java Virtual Machine Specification*, Version, Alpha 1 Release, Technical report, Computational Logic, Inc, http://www.cli.com/software/djvm/html-0.5/djvm-report.html, 1996.
3. Schlumberger, Inc.: *Cyberflex 2.0 Multi 8K Programmer's Guide,* http://www.cyberflex.austin.et.slb.com/cyberflex/docs/docs-page3.htm, 1997.
4. D. Dean, E. W. Felten, and D. S. Wallach*: Java Security: From HotJava to Netscape and Beyond*, IEEE Symposium on Security and Privacy, Oakland, California, May 1996.
5. Sun Microsystems, Inc.: *Java Card 2.0 Language Subset and Virtual Machine Specification,* Revision 1.0 Final, http://java.sun.com:80/products/javacard, 1997.
6. Sun Microsystems, Inc.: *Java Card 2.0 Application Programming Interfaces,* Revision 1.0 Final, Revision 1.0 Final, http://java.sun.com:80/products/javacard, 1997.
7. J. Posegga and H. Vogt*: Offline Verification for Java Card Byte Code Using a Model Checker, 1998 (submitted to ESORICS-98).*
8. J. Posegga: *Die Sicherheitsaspekte von Java,* Informatik-Spektrum vol. 21, no. 1, pp. 16-21, Springer, 1998.
9. M. Kaiserswerth and J. Posegga: *Java Chipkarten*, Informatik-Spektrum vol. 21, no. 1, pp. 27-28, Springer, 1998.

10. F. Yellin: *Low Level Security in Java*, WWW4, http://www.w3.org/pub/Conferences/WWW4/Papers/197/40.html, 1995.