Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Σχολή Θετικών Επιστημών
Τμήμα Πληροφορικής

Στατική Ανάλυση Εφαρμογών Έξυπνων Καρτών Java Card

Επιβλέπων Καθηγητής: Κατσαρός Παναγιώτης

Επιμέλεια: Αλμαλιώτης Βασίλειος

Θεσσαλονίκη, Οκτώβριος 2008

Στατική Ανάλυση Εφαρμογών Έξυπνων Καρτών
Java Card

Επιβλέπων Καθηγητής: Κατσαρός Παναγιώτης

Επιμέλεια: Αλμαλιώτης Βασίλειος

Αριθμός Ειδικού Μητρώου: 874

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Τμήμα Πληροφορικής

Θεσσαλονίκη 2008

Περιεχόμενα

Εισαγωγή

Καθημερινά εκτελούνται χιλιάδες αγορές μέσω των πιστωτικών καρτών. Θα ήταν πολύ πιο δύσκολο και συνάμα λιγότερο ασφαλές εάν όλες αυτές οι συναλλαγές δε χρησιμοποιούσαν κάρτες αλλά μετρητά χαρτονομίσματα. Η ασφάλεια και η φορητότητα των έξυπνων καρτών παρέχει ένα αξιόπιστο και αποτελεσματικό τρόπο ηλεκτρονικών συναλλαγών αλλά και μία πλατφόρμα ανάπτυξης μεγάλου εύρους εφαρμογών. Έτσι σήμερα οι έξυπνες κάρτες χρησιμοποιούνται για:

- πιστοποίηση αυθεντικότητας και τον έλεγχο πρόσβασης στο διαδίκτυο,
- αποθήκευση ιατρικού ιστορικού,
- μεταφορά χρημάτων, «ηλεκτρονικό πορτοφόλι»
- έλεγχο πρόσβασης σε χώρους
- βελτίωση της ασφάλειας της ασύρματης τηλεφωνίας
- δραστηριότητες φοιτητών όπως βιβλιοθήκη, πάσο, δήλωση μαθημάτων
- δραστηριότητες των εργαζομένων μιας επιχείρησης

Σε αυτήν την εργασία παρουσιάζονται οι ιδέες για στατική ανάλυση προγραμμάτων σε έξυπνες κάρτες, με σκοπό την εύρεση αδυναμιών τους. Έτσι είναι σημαντική μια εκτενής αναφορά στις έξυπνες κάρτες και την τεχνολογία τους καθώς και στην διαδικασία της ανάπτυξης προγραμμάτων πάνω σε αυτές, προτού αναφερθεί η ανάλυση που είναι δυνατών να γίνει πάνω σε αυτά. Στο πρώτο μέρος, παρουσιάζονται τα βασικά χαρακτηριστικά των έξυπνων καρτών τόσο από την πλευρά του υλικού όσο και από του λογισμικού. Ακολουθεί η τεχνολογία Java Card, όπου αναλύονται διεξοδικά η αρχιτεκτονική της και ο τρόπος επικοινωνίας των εφαρμογών στην περίπτωση ενός APDU καθώς και ενός RMI λογισμικού. Τέλος, εξηγείται ο τρόπος ανάπτυξης μιας Java Card εφαρμογής και ο τρόπος εκτέλεσης της εφαρμογής που αναπτύχθηκε. Στο δεύτερο μέρος παρουσιάζεται η υλοποίηση ορισμένων ανιχνευτών, στο λογισμικό ανοιχτού κώδικα FindBugs, οι οποίοι έχουν την δυνατότητα του ελέγχου κάποιων ιδιοτήτων των προγραμμάτων αυτών που εισάγονται στην κάρτα.

Μέρος Πρώτο:
Οι έξυπνες κάρτες
και η τεχνολογία τους

Έξυπνες Κάρτες

Γενικά

Μία «έξυπνη» κάρτα, smart card ή chip card ή integrated circuit card(ICC),είναι ένα ολοκληρωμένο κύκλωμα προσαρμοσμένο σε μία μικρού μεγέθους κάρτα. Η ιδέα της ενσωμάτωσης ενός ολοκληρωμένου κυκλώματος σε μία πλαστική κάρτα διατυπώθηκε από τους Γερμανούς εφευρέτες Jürgen Dethloff και Helmut Grotrupp το 1968. Η τεχνολογία των έξυπνων καρτών ως βιομηχανικό πρότυπο ορίστηκε και ελέγχεται από τον Joint Technical Committee 1 (JTC1) του International Standards Organization (ISO) and the International Electronic Committee (IEC). Με τη σειρά ISO/IEC 7816 η οποία ξεκίνησε το 1987 καθορίζονται διάφορα χαρακτηριστικά των καρτών αυτών όπως φυσικά χαρακτηριστικά, εντολές, τρόπος σύνδεσης, πρωτόκολλα συναλλαγών, ηλεκτρονικά σήματα, ασφάλεια αρχιτεκτονικής, αναγνωριστικό εφαρμογής τα οποία θα αναλυθούν παρακάτω. Υπάρχουν τρεις βασικές κατηγορίες ICC:

- οι κάρτες μνήμης, περιλαμβάνουν ένα κύκλωμα μνήμης και προγραμματίζονται μία φορά από την κατασκευάστρια εταιρία,
- οι οπτικές κάρτες μνήμης, περιλαμβάνουν κύκλωμα μνήμης και προβλέπεται στο μέλλον να διαθέτουν και μικροεπεξεργαστή
- οι κάρτες με μικροεπεξεργαστή, περιλαμβάνουν ένα κύκλωμα μνήμης και ένα μικροεπεξεργαστή

Οι πρώτες έχουν την ικανότητα αποθήκευσης δεδομένων χωρίς όμως να μπορούν να τα επεξεργαστούν αφού δε διαθέτουν μικροεπεξεργαστή. Για το λόγο αυτό λέγεται ότι δεν είναι «έξυπνες». Περιέχουν ένα κύκλωμα μνήμης ή ένα κύκλωμα με μνήμη και μη-προγραμματιζόμενη λογική. Η επεξεργασία των δεδομένων γίνεται με λίγες προ-προγραμματισμένες εντολές με αποτέλεσμα να υποστηρίζεται μικρός αριθμός λειτουργιών. Σημαντικό μειονέκτημα αποτελεί το γεγονός ότι δεν ξαναπρογραμματίζονται άρα δεν ξαναχρησιμοποιούνται. Για παράδειγμα, μια τηλεφωνική κάρτα προπληρωμένου χρόνου που δεν έχει πλέον υπόλοιπο είναι εντελώς άχρηστη. Ήταν οι πρώτες που κυκλοφόρησαν σε μεγάλες ποσότητες. Η πρώτη πατέντα για αυτό το είδος κάρτας ανήκει στον Roland Moreno και χρονολογείται το 1977. Σήμερα χρησιμοποιούνται κυρίως για αποθήκευση δεδομένων, προσωπική χρήση, αλλά

και σαν τηλεφωνικές κάρτες προπληρωμένου χρόνου. Ανάλογα με τις λειτουργίες που υποστηρίζουν μπορούμε να τις διακρίνουμε σε κάρτες ανάγνωσης/εγγραφής (Read/Write) και σε ανάγνωσης/διαγραφής (Read/Erase only)

Οι οπτικές κάρτες μνήμης, είναι εξωτερικά σαν κάρτες που έχουν κολλημένες στη μια πλευρά τους ένα οπτικό δίσκο(cd). Το κόστος μιας τέτοιας κάρτας είναι συγκρίσιμο με αυτές που διαθέτουν μικροεπεξεργαστή αλλά το κόστος της συσκευής ανάγνωσης (card reader) είναι πολύ μεγαλύτερο. Ακόμα δεν υπάρχουν κοινά αποδεκτά πρότυπα για την επικοινωνία κάρτας τερματικού.

Οι κάρτες με μικροεπεξεργαστή προσφέρουν αυξημένη ασφάλεια και συνάμα υποστήριξη πολλών διαφορετικών λειτουργιών. Χάρη στον μικροεπεξεργαστή τα δεδομένα δεν είναι ποτέ απευθείας διαθέσιμα σε εξωτερικές εφαρμογές. Εφευρέθηκαν από τον Michel Ugon το 1977. Έχουν την ικανότητα τόσο αποθήκευσης όσο και επεξεργασίας δεδομένων. Η επεξεργασία γίνεται σύμφωνα με τον κώδικα που υπάρχει στην κάρτα, τον οποίο μπορούμε να επαναφορτώσουμε, διαγράψουμε, προσθέσουμε αλλάζοντας έτσι τη λειτουργία της κάρτας. Λόγο του υψηλού επιπέδου ασφάλειας που προσφέρουν χρησιμοποιούνται συχνά για τη μεταφορά χρημάτων, έλεγχο πρόσβασης και γενικότερα όπου η ασφάλεια των δεδομένων και διαφύλαξη προσωπικών δεδομένων αποτελούν προτεραιότητα.

Γενικά με τον όρο έξυπνη κάρτα αναφερόμαστε σε όλα τα παραπάνω είδη κάρτας. Παρόλα αυτά, μερικοί χρησιμοποιούν τον όρο μόνο όταν υπάρχει μικροεπεξεργαστής επειδή όπως χαρακτηριστικά λένε «η εξυπνάδα πηγάζει από την επεξεργασία». Ένας κοινά αποδεκτός όρος για την περιγραφή και των τριών κατηγοριών που παρουσιάστηκαν είναι κάρτα με κύκλωμα (chip card).

Contact και Contactless

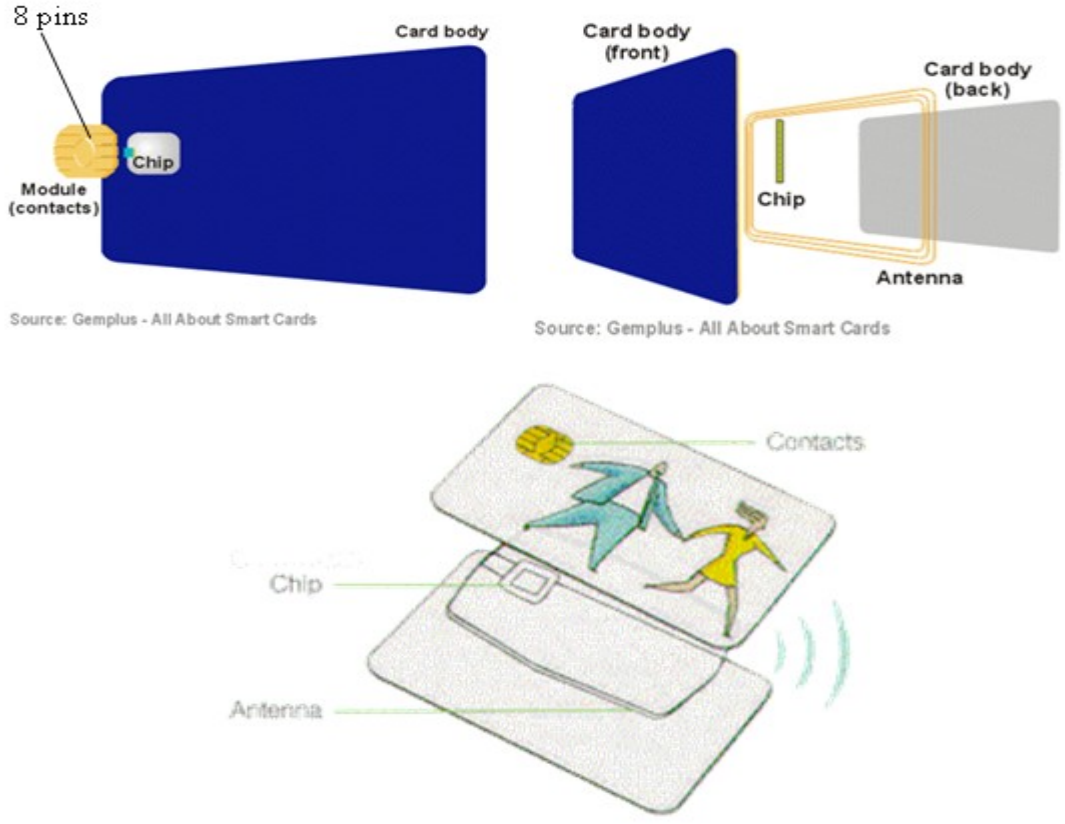
Μία κάρτα δε μπορεί να λειτουργήσει αυτόνομα. Δηλαδή, είναι απαραίτητη η επικοινωνία της με άλλες συσκευές για την ανταλλαγή δεδομένων εισόδου και εξόδου αλλά και για την παροχή ενέργειας. Οι συσκευές που «συνδέονται» με τις κάρτες λέγονται CAD (Card Acceptance Device) και μπορεί να είναι είτε card reader μέσω ενός υπολογιστή είτε ένα τερματικό το οποίο λειτουργεί αυτόνομα. Ανάλογα με τη

συναλλαγής στην οποία σκοπεύουμε να χρησιμοποιήσουμε μια κάρτα μπορούμε να επιλέξουμε έναν από τους παρακάτω τύπους

Contact, λέγεται η κάρτα η οποία για να συνδεθεί με τη συσκευή ανάγνωσης απαιτεί φυσική επαφή μέσω των 8 pins που διαθέτει. Οι κάρτες αυτές έχουν περιορισμένο πάχος αφού θα πρέπει να εφαρμόζουν στην υποδοχή ανάγνωσης.

Στην περίπτωση μιας contactless κάρτας η επικοινωνία γίνεται ασύρματα με ηλεκτρομαγνητικά πεδία. Η ενέργεια μπορεί να παρέχεται με μια μικρή μπαταρία είτε μέσω της κεραίας. Αυτές οι κάρτες δε διαθέτουν pins αλλά κεραία η οποία απαιτεί η μέγιστη απόσταση να είναι περίπου δέκα μέτρα ενώ πριν από πέντε χρόνια ήταν περίπου εξήντα εκατοστά. Αυτές οι κάρτες χρησιμοποιούνται σε περιπτώσεις όπου θέλουμε γρήγορες συναλλαγές όπως δημόσιο σύστημα μεταφορών και έλεγχος πρόσβασης.

Τέλος, οι combination cards είναι συνδυασμός των δύο παραπάνω κατηγοριών αφού υποστηρίζουν σύνδεση τόσο με φυσική επαφή όσο και με ραδιοσυχνότητα. Στο παρακάτω σχήμα φαίνονται μια contact (αριστερά) και μια contactless (δεξιά).



Σχήμα 1 Αριστερά contact, δεξιά contactless κάρτα, κάτω combination

Smart Card Hardware

Μια έξυπνες κάρτες διαθέτουν κάποια σημεία επαφής με την CAD, στην επιφάνεια του πλαστικού υποστρώματος, μια κεντρική μονάδα επεξεργασίας, αρκετά είδη μνήμης και σε κάποιες περιπτώσεις ένα ειδικό επεξεργαστή για μαθηματικούς υπολογισμούς.

Σημεία Επαφής (Contact Pins)

Κάθε κάρτα έχει οκτώ σημεία επαφής με τα οποία συνδέεται στον αναγνώστη καρτών(card reader)(Σχήμα 2). Το κάθε ένα επιτελεί την παρακάτω λειτουργία:

- Vcc

  Παροχή ενέργειας στο κύκλωμα με τιμές 3 ή 5 Volts

- RST

  Χρησιμεύει για την αποστολή σήματος με σκοπό την επαναφορά αρχικών συνθηκών στο μικροεπεξεργαστή (warm reset). Ένας άλλος τρόπος reset είναι διακοπή παροχής ρεύματος και μετά ξανά παροχή, on off, ή η εξαγωγή της κάρτας και μετά ξανά εισαγωγή. Οι δύο τελευταίες περιπτώσεις λέγονται cold reset.

- CLK

  Επειδή ο επεξεργαστείς δεν διαθέτουν εσωτερική γεννήτρια ρολογιού το CLK στέλνει το σήμα του εξωτερικού ρολογιού στο εσωτερικό ρολόι.

- GND

  Γείωση

- Vpp

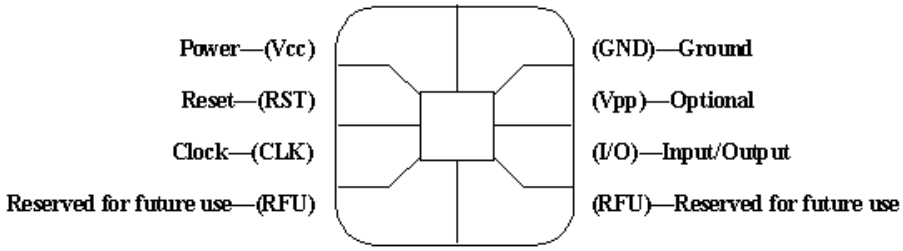
  Αυτό το σημείο επαφής είναι προαιρετικό και δεν χρησιμοποιείται στις καινούργιες κάρτες. Σε κάποιες παλιές κάρτες ήταν απαραίτητο για τον προγραμματισμό της μνήμης EEPROM

- I/O

  Από εδώ γίνεται η μεταφορά τόσο των δεδομένων όσο και των εντολών από και προς την κάρτα με μη ταυτόχρονο τρόπο (half-duplex). Δηλαδή είναι δυνατή η μετάδοση εντολών ή δεδομένων σε μία μόνο κατεύθυνση τη φορά

- RFU

Προς το παρών δε χρησιμοποιούνται. Υπάρχουν για μελλοντική χρήση.



Σχήμα 2: Τα σημεία επαφής μιας contact smart card

Μονάδες Επεξεργασίας

Οι έξυπνες κάρτες στο παρελθόν διέθεταν 8-bit μικροελεγκτή (microcontroller) και χρησιμοποιούσαν συνήθως για ρεπερτόριο εντολών το Motorola 6805 ή το intel 8051. Τα τελευταία χρόνια εμφανίστηκε το μειωμένο ρεπερτόριο εντολών (RISC) και 16 ή 32 bit μικροελεγκτής. Ακόμα, οι κάρτες οι οποίες προορίζονται για χρήση σε εφαρμογές ασφάλειας είναι εφοδιασμένες με ένα ειδικό συνεπεξεργαστή(coprocessor) για πολύπλοκες αριθμητικές πράξεις που απαιτούνται από κρυπτογραφικούς αλγόριθμους όπως RSA.

Μνήμη

Συνήθως σε μία έξυπνη κάρτα συναντάμε τρία είδη μνήμης ROM, EEPROM και RAM. Πιο ακριβή είναι η EEPROM και πιο φθηνή η ROM. Η RAM είναι αρκετά σπάνια σε έξυπνες κάρτες. Παρακάτω παρουσιάζονται τα βασικά χαρακτηριστικά καθώς και η χρησιμότητα της κάθε μνήμης στις έξυπνες κάρτες:

- ROM(Read only Memory)

  Χρησιμοποιείται για την αποθήκευση προγραμμάτων στην κάρτα. Δεν απαιτείται ενέργεια για τη διατήρηση των δεδομένων. Παρόλα αυτά δεν μπορούμε να ξαναγράψουμε κάτι στη μνήμη μετά την κατασκευή της κάρτας. Η ROM μιας έξυπνης κάρτας περιέχει ρουτίνες του λειτουργικού συστήματος, μόνιμα δεδομένα και της εφαρμογές του χρήστη.
- EEPROM(Electrical Erasable Programmable Read Only Memory)

Όπως και η ROM έχει την ικανότητα να διατηρεί δεδομένα χωρίς να είναι απαραίτητη η παρουσία ενέργειας. Η διάφορα των δύο αυτών τύπων είναι ότι η EEPROM μπορεί να μεταβληθεί όσες φορές θέλουμε. Έτσι χρησιμοποιείται για την αποθήκευση δεδομένων όπως ο σκληρός δίσκος στον προσωπικό υπολογιστή. Οι εφαρμογές χρήστη μπορούν να γραφτούν στη μνήμη αυτή μετά την κατασκευή της κάρτας. Μία EEPROM σε smart card μπορεί να δεχτεί τουλάχιστον 100,000 κύκλους εγγραφής και να διατηρήσει τα δεδομένα για δέκα χρόνια. Η ταχύτητα ανάγνωσης της είναι περίπου τόσο όσο και της RAM, αρκετά γρήγορη, αλλά η ταχύτητα εγγραφής είναι 1,000 φορές πιο αργή από αυτή της RAM.

Τα τελευταία χρόνια έχουν εμφανιστεί και άλλες τεχνολογίες μνήμης όπως η flash. Αυτή προτιμάται για την αποθήκευση προγραμμάτων ή μεγάλων μπλοκ δεδομένων.

Smart Card Operating System

Το λειτουργικό σύστημα μιας έξυπνης κάρτας είναι ένα κομμάτι λογισμικού το οποίο επικοινωνεί με το hardware, τα κυκλώματα της κάρτας, παρέχοντας βασικές λειτουργίες όπως ασφαλή πρόσβαση στα δεδομένα, πιστοποίηση, κρυπτογράφηση. Χωρίς αυτό κάθε εφαρμογή θα έπρεπε να επικοινωνεί απευθείας με το υλικό με αποτέλεσμα η εγγραφή κώδικα για εφαρμογές κάρτας να απαιτούσαν προγραμματισμό πολύ χαμηλού επιπέδου.

Επειδή το λειτουργικό σύστημα φορτώνεται στη μνήμη ROM μιας έξυπνης κάρτας, αυτό θα πρέπει οπωσδήποτε να μην έχει λάθη. Έτσι αν διαπιστωθεί κάποιο σημαντικό λάθος θα πρέπει να αντικατασταθούν όλες οι κάρτες που έχουν εκδοθεί και η καταστροφή όλων των ICs σε αυτήν τη συγκεκριμένη σειρά παραγωγής. Για να αποφευχθεί αυτό πολλά λειτουργικά συστήματα επιτρέπουν τη χρησιμοποίηση επιπρόσθετων προγραμμάτων διόρθωσης λαθών(patches), έτσι ώστε τα μεγαλύτερα μέρη τους να μπορούν να φορτωθούν μέσα στην EEPROM, εάν πληρούνται οι απαραίτητες συνθήκες ασφαλείας. Παρόλα αυτά, τα λειτουργικά συστήματα των έξυπνων καρτών θα πρέπει να αναπτύσσονται και να ελέγχονται πιο προσεκτικά από ότι τα συμβατικά.

Τα λειτουργικά συστήματα πολλαπλών εφαρμογών υλοποιούνται σε ICs, οι οποίες παρέχουν ικανοποιητική ασφάλεια υλικού που εγγυάται ξεχωριστές περιοχές

μνήμης, μπορούν μάλιστα να επιτρέψουν τη φόρτωση μιας καινούργιας εφαρμογής μετά την έκδοση της κάρτας. Αυτό υποστηρίζεται τόσο από το Java Card και το MULTOS όσο και από το Windows for Smart Cards, τα οποία είναι τα πιο σημαντικά λειτουργικά συστήματα έξυπνων καρτών.

Υπάρχουν δύο κατηγορίες λειτουργικών συστημάτων:

- Καθορισμένης Δομής Αρχείων (Fixed File Structure)

  Αυτό το σύστημα μεταχειρίζεται την κάρτα σαν μια ασφαλή μονάδα υπολογισμού και αποθήκευσης δεδομένων. Τα αρχεία αλλά και τα δικαιώματα ορίζονται από τον εκδότη. Για αυτό το λόγο είναι ιδανικό για εφαρμογές που δεν προβλέπεται να χρειαστεί αλλαγή της λειτουργίας τους ή αναβάθμιση στο κοντινό μέλλον.

- Δυναμικό Σύστημα Εφαρμογών (Dynamic Application System)

  Αυτή η κατηγορία λειτουργικών επιτρέπει ανάπτυξη, τον έλεγχο εφαρμογών οι οποίες μπορούν να συνεργάζονται με ασφάλεια. Εδώ ανήκουν τα δύο πιο γνωστά λειτουργικά συστήματα για έξυπνες κάρτες το MULTOS και το JAVA card. Υπάρχει σαφέστερος διαχωρισμός του λειτουργικού από τις εφαρμογές και έτσι είναι ευκολότερη η αναβάθμιση των εφαρμογών. Για παράδειγμα η αναβάθμιση της κάρτας SIM για κινητά GSM πραγματοποιείται κατεβάζοντας τη νέα έκδοση αλλάζοντας έτσι τις λειτουργίες της κάρτας δυναμικά.

Τεχνολογία Java Card

Εισαγωγή

Αρχικά, το λογισμικό των έξυπνων καρτών ήταν γραμμένο ειδικά για το συγκεκριμένο μικροεπεξεργαστή της κάθε κάρτας. Αυτό σημαίνει ότι είναι πολύ δύσκολη η συνεργασία εφαρμογών που προορίζονται για διαφορετικές κάρτες. Μια λύση στο πρόβλημα αυτό, αποτελεί η τεχνολογία που μας προσφέρει η sun για την ανάπτυξη εφαρμογών στις έξυπνες κάρτες με το υποσύνολο της γλώσσας java, την java card. Αυτή χρησιμοποιείται και για τον προγραμματισμό smart buttons αλλά και USB tokens.

Σχήμα 3 Αριστερά smart button, δεξιά USB token

Αρχιτεκτονική Java Card

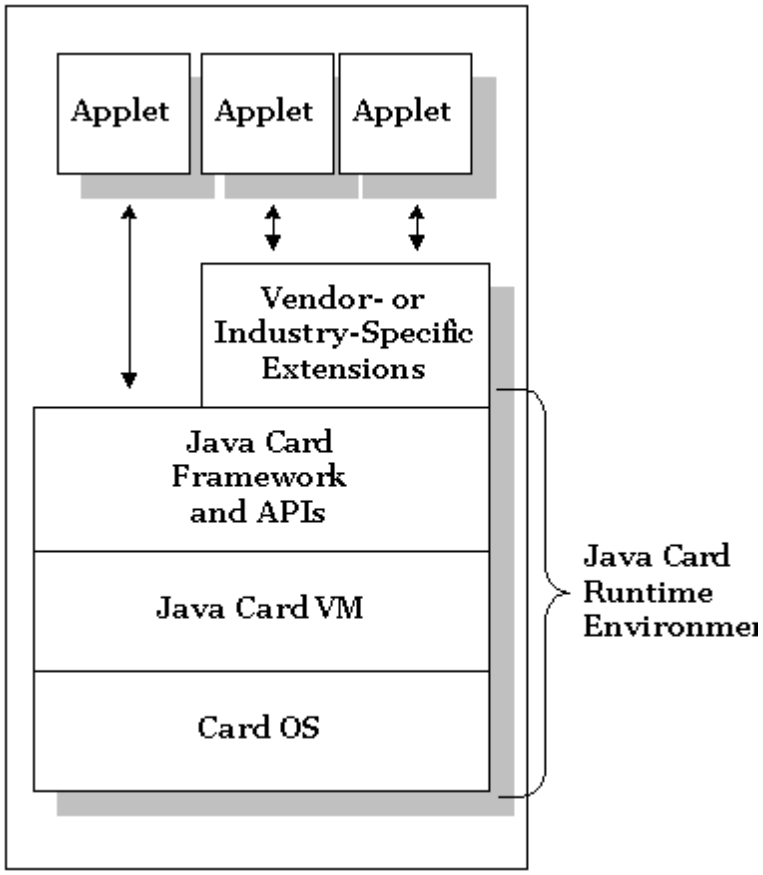
Τα προγράμματα τα οποία είναι ικανά να φορτωθούν σε μια java card είναι όλα java card applets.Τα applets αυτά επικοινωνούν με το Java Card Runtime Environment (JCRE). Υποστηρίζεται επίσης η δυνατότητα ανάπτυξης ενός ενδιάμεσου επιπέδου το οποίο θα παρεμβάλλεται στην επικοινωνία των appletse το JCRE και λέγεται Vendor- or Industry-Specific Extensions. Το JCRE αποτελείται από:

- Java Card Framework and APIs

Περιλαμβάνει τις κλάσεις και τα πακέτα που είναι απαραίτητα για τον προγραμματισμό μιας java card. Είναι το πιο «υψηλό» επιπέδου και έρχεται σε επικοινωνία με τα applets πριν από όλα. Τέλος, στέλνει τις εντολές στο Java Card Virtual Machine.

- Java Card Virtual Machine

Ορίζει ένα υποσύνολο της γλώσσας Java, τη Java Card, και της Java Virtual Machine, τη Java Card Virtual Machine. Χωρίζεται σε δύο μέρη εκ των οποίων το ένα βρίσκεται εκτός κάρτας, java card converter, και το άλλο εντός. Το δεύτερο περιλαμβάνει τον Java Card Inerpreter ο οποίος εκτελεί εντολές Java Card Bytecode, ελέγχει τη δέσμευση μνήμης και τη δημιουργία



Αρχιτεκτονική Java Card

αντικειμένων, παίζει κύριο ρόλο για την ασφάλεια κατά το χρόνο εκτέλεσης. Ο Java Card Converter βρίσκεται εκτός κάρτας και η κύρια λειτουργία του είναι η μετατροπή των αρχείων που θέλουμε να φορτώσουμε στην κάρτα, παράδειγμα μια εφαρμογή, σε μορφή τέτοια ώστε να επιτρέπεται η φόρτωση και η εκτέλεση της. Με άλλα λόγια μετατρέπει τα αρχεία .class, τα οποία παράγονται από τον πηγαίο κώδικα (source code) μέσω ενός Java compiler, σε ένα αρχείο CAP. Κατά τη διαδικασία της μετατροπής, ο Java Card Converter εκτελεί κάποιες ενέργειες τις οποίες Java Virtual Machine εκτελεί κατά τη φόρτωση κλάσεων (class loading):

- Πιστοποιεί ότι οι java κλάσεις είναι σωστά σχηματισμένες.
- Ελέγχει για τυχών παραβιάσεις της Java Card.
- Αρχικοποιεί τις static μεταβλητές.
- Αναλύει συμβολικές αναφορές σε κλάσεις, μεθόδους και πεδία σε μια συμπαγή μορφή την οποία η κάρτα μπορεί να διαχειριστεί πιο αποτελεσματικά.
- Βελτιστοποιεί τον bytecode μέσω της εκμετάλλευσης πληροφοριών που λαμβάνονται κατά τη φόρτωση των κλάσεων και κατά τη διάρκεια της σύνδεσης (linking time).
- Εκχωρεί μνήμη και δημιουργεί δομές δεδομένων της Virtual Machine για την αναπαράσταση κλάσεων.

❖ Card OS

Το λειτουργικό σύστημα της κάρτας το οποίο υλοποιεί τις βασικές της λειτουργίες όπως έχει αναλυθεί πιο πάνω.

Διαφορές Java – Java Card

Εξαιτίας του μικρού μεγέθους της μνήμης, η πλατφόρμα της Java Card, υποστηρίζει μόνο ένα προσεκτικά επιλεγμένο, προσαρμοσμένο υποσύνολο χαρακτηριστικών της γλώσσας Java. Αυτό το υποσύνολο περιλαμβάνει χαρακτηριστικά, τα οποία είναι κατάλληλα για την ανάπτυξη προγραμμάτων έξυπνων καρτών και άλλων μικρών συσκευών, ενώ παράλληλα διατηρεί τις ικανότητες του αντικειμενοστραφή προγραμματισμού της Java. Πολλές προηγμένες έξυπνες κάρτες της java παρέχουν μηχανισμό συλλογής άχρηστων αντικειμένων (garbage collection) που καθιστούν δυνατή

τη διαγραφή των αντικειμένων. Στο παρακάτω σχήμα παρουσιάζονται χαρακτηριστικά της Java τα οποία είτε υποστηρίζονται (αριστερή στήλη) είτε όχι (δεξιά στήλη) από την Java Card.

Πίνακας 1 Διαφορές Java - Java Card

Χαρακτηριστικά της Java	
Υποστηρίζονται	**Δεν Υποστηρίζονται**
Μικροί στοιχειώδης τύποι δεδομένων: Boolean, byte, short	Μεγάλοι στοιχειώδης τύποι δεδομένων: long, double, float. Χαρακτήρες και αλφαριθμητικά (strings)
Μονοδιάστατοι πίνακες	Πολυδιάστατοι πίνακες
Πακέτα της Java(Java packages), Κλάσεις(classes),Διασυνδέσεις (Interfaces) και εξαιρέσεις (exceptions)	Δυναμική φόρτωση κλάσεων (Dynamic class loading) Διαχειριστής ασφαλείας (Security manager)
Αντικειμενοστραφή χαρακτηριστικά της Java: Κληρονομικότητα, virtual methods, overloaded and dynamic object creation, εμβέλεια πρόσβασης (access scope), κανόνες δέσμευσης (binding rules)	Finalization Object serialization Νήματα (Threads)
Η λέξη κλειδί int και ο 32-bit integer τύπος δεδομένων υποστηρίζονται προαιρετικά	Object cloning

Java Card Applets

Εισαγωγή

Τα προγράμματα τα οποία είναι ικανά να φορτωθούν σε μια java card είναι όλα java card applets. Λέγονται applets επειδή μοιάζουν σε κάποιο βαθμό με τα applets της java. Η κύρια διαφορά με τα applets της java είναι ότι εγκαθίστανται και αρχικοποιούνται μία φορά και η εκτέλεση τους δεν τερματίζεται αλλά απλά διακόπτεται. Υπάρχουν δύο μέθοδοι για να υλοποιηθεί ένα java card applet. Η μία έχει τη λογική του μοντέλου αιτήσεων-απαντήσεων, στην οποία μια client εφαρμογή στέλνει αιτήματα και λαμβάνει αποκρίσεις από την κάρτα (πλέον παραδοσιακή μέθοδος η οποία έχει βάση τα APDU's) και η δεύτερη, η οποία είναι και η πιο μοντέρνα, υλοποιεί τη φιλοσοφία της κλήσης απομακρυσμένων μεθόδων (RMI). Το RMI (Remote Method Invocation) όπως εισάγεται από την Java, είναι η μέθοδος που μας επιτρέπει να πραγματοποιούμε απομακρυσμένες κλήσεις, μεταξύ διαφορετικών προγραμμάτων. Οι κάρτες που είναι προορισμένες για java card μπορούν να υποστηρίξουν πολλαπλές εφαρμογές στην ίδια κάρτα. Μέσα στην κάρτα μπορούν να συνυπάρχουν και να αλληλεπιδρούν με ασφάλεια, εφαρμογές και των δύο μεθόδων.

Οι εφαρμογές της κάρτας, java card applets, επικοινωνούν με εφαρμογές που βρίσκονται εκτός κάρτας, στο τερματικό ή τον υπολογιστή, ανάλογα με το CAD που θα χρησιμοποιήσουμε. Έτσι, στην περίπτωση του APDU, τα applets παίζουν το ρόλο του εξυπηρέτη (server) που απαντάει σε αιτήσεις του πελάτη (client), εφαρμογές εκτός κάρτας. Στο μοντέλο του RMI δεν φαίνεται ξεκάθαρα η λογική πελάτη – εξυπηρέτη (client - server). Οι εφαρμογές εντός και εκτός κάρτας επικοινωνούν μεταξύ τους με κλήσεις συναρτήσεων. Στη συνέχεια, θα παρουσιαστούν οι εφαρμογές εντός και εκτός κάρτας καθώς και ο τρόπος επικοινωνίας τους ανάλογα με την τεχνολογία υλοποίησης που χρησιμοποιείται, Application Protocol Data Unit (A.P.D.U.) ή Remote Method Invocation (R.M.I.)

Applet AID

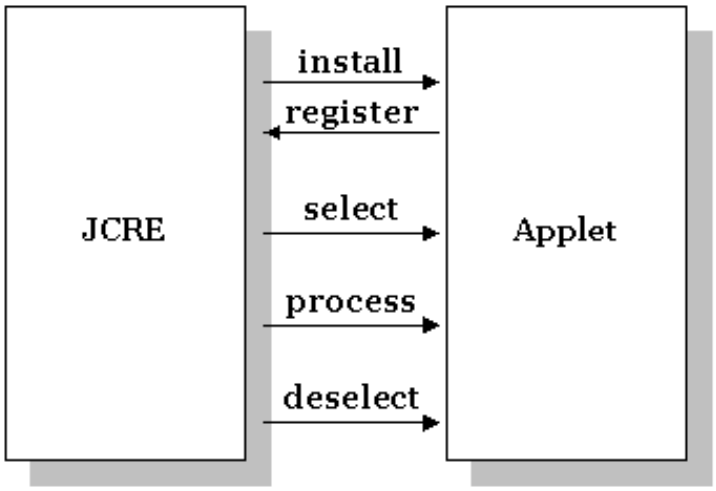
Σε κάθε εφαρμογή και σε κάθε πακέτο ανατίθεται ένας αριθμός AID. Αυτή η συμβολοσειρά από bytes είναι το αναγνωριστικό για κάθε πακέτο και εφαρμογή και χρησιμεύει για τη διάκριση τους μεταξύ άλλων εφαρμογών, που μπορεί να συνυπάρχουν μέσα στην κάρτα ή για την επικοινωνία με αυτές. Τα πρώτα 5 bytes αυτού λειτουργούν σαν αναγνωριστικά του κατασκευαστή του λογισμικού και ονομάζονται RID και τα υπόλοιπα 0 μέχρι 11, τα οποία ονομάζονται PIX, ορίζονται από τον ελεύθερα από τον κατασκευαστή, και λειτουργούν σαν αναγνωριστικά των πακέτων και των εφαρμογών. Αυτή η σύμβαση έχει οριστεί από τον οργανισμό ISO στο πρότυπο ISO 7816-5. Οι εφαρμογές που ανήκουν σε κάποιο πακέτο έχουν το ίδιο AID με το πακέτο με τη διαφορά ότι έχουν ακόμα ένα byte που διαφοροποιεί τις εφαρμογές μέσα στο πακέτο. Για αυτό λέγεται ότι το Applets κληρονομούν το AID του πακέτου. Παράδειγμα της δομής ενός Applet AID φαίνεται στο παρακάτω σχήμα.

<table>
<tr><td colspan="3">Applet AID</td></tr>
<tr><td colspan="2">Package AID</td><td></td></tr>
<tr><td>0xA0 0x00 0x00 0x00 0x62</td><td>0x03 0x01 0x0C 0x08</td><td>0x01</td></tr>
<tr><td>RID</td><td colspan="2">PIX</td></tr>
</table>

Σχήμα 4 Η μορφή του AID

Επικοινωνία Applet – Java Card Runtime Environment (JCRE)

Κάθε Applet πρέπει υποχρεωτικά να υλοποιεί κάποιες βασικές λειτουργίες για τη σωστή επικοινωνία του με το JCRE. Η επικοινωνία αυτή βασίζεται σε πέντε βασικές λειτουργίες από τις οποίες προκύπτουν ορισμένα σημεία εισόδου (multiple entry points) σε ένα Applet, δηλαδή συγκεκριμένοι τρόποι επικοινωνίας αυτού με τον έξω κόσμο. Παρακάτω παρουσιάζονται οι μέθοδοι αυτοί καθώς και ποιος έχει δικαίωμα κλήσης της κάθε μεθόδου.



Σχήμα 5 Επικοινωνία JCRE - Applet

- install() – Εγκατάσταση αντικειμένου

Η μέθοδος αυτή εκτελείται μόνο μία φορά στη ζωή του applet και συχνά περιέχει κάποια ορίσματα αρχικοποίησης. Κύρια της λειτουργία είναι η δημιουργία ενός στιγμιότυπου, της κλάσης του apple,t και η καταχώρηση του στο JCRE. Η καταχώρηση γίνεται με την κλήσης της μεθόδου register(), η οποία είναι υποχρεωτική να περιλαμβάνεται σε κάθε install(). Σε αντίθετη περίπτωση η εγκατάσταση αποτυγχάνει. Καλείται από το JCRE ενώ το σώμα της υλοποιείται στο applet.

- register() – Καταχώρηση στιγμιότυπου

Η μέθοδος αυτή καλείται από το applet με σκοπό την καταχώρηση του στο JCRE. Ειδικότερα, κατά την εκτέλεση της μεθόδου install() δημιουργείται ένα στιγμιότυπου της κλάσης του applet που επιθυμούμε να εγκαταστήσουμε. Αυτό το στιγμιότυπο, καταχωρείται στο Java Card Runtime Environment(JCRE). Η κλήση της μεθόδου γίνεται από το σώμα της μεθόδου install() και καλείται από το applet.

- select – Επιλογή της εφαρμογής

Η διαδικασία αυτή είναι απαραίτητη κάθε φορά που το applet πρόκειται να εκτελέσει κάποια λειτουργία. Δηλαδή πρέπει αρχικά να επιλεγεί ώστε να γνωρίζει το JCRE σε ποιο applet αναφερόμαστε. Έτσι κατά την επιλογή ενός applet ειδοποιούμε το JCRE με κατάλληλο μήνυμα ότι θέλουμε να επιλέξουμε ένα applet. Μέσα στο μήνυμα αυτό περιλαμβάνεται το AID του applet. Αν το JCRE περιλαμβάνει αυτό το AID στα

καταχωρημένα στιγμιότυπα, το applet υπάρχει και έχει εγκατασταθεί κανονικά, καλεί τη select() μέθοδο επιλέγοντας έτσι το applet. Τώρα το applet μπορεί να εκτελέσει τις λειτουργίες που υποστηρίζει. Μόνο ένα applet μπορεί να είναι επιλεγμένο κάθε φορά.

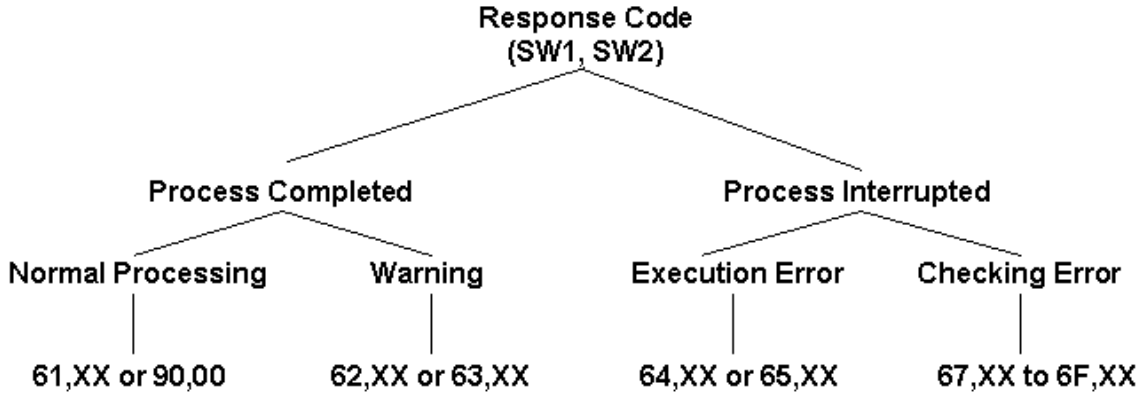
- process – Εκτέλεση εντολής εντός εφαρμογής

Με αυτή τη μέθοδο το applet προσφέρει τις λειτουργίες του. Αφού έχει γίνει σωστά η διαδικασία της εγκατάστασης και της επιλογής ενός applet επιθυμούμαι να εκτελέσουμε κάποια από τις λειτουργίες που υποστηρίζει. Με την αποστολή και πάλι του κατάλληλου μηνύματος στην κάρτα το JCRE εκτελεί τη μέθοδο process() με τα κατάλληλα ορίσματα που τις αποστείλαμε στο μήνυμα. Η process(), που συνήθως έχει τη μορφή «switch case» λόγο των πολλών λειτουργιών που συνήθως επιτελούν τα applets, εκτελεί την κατάλληλη συνάρτηση.

- deselect – Εγκατάλειψη της εφαρμογής

Αντίθετη με τη μέθοδο select(), Εκτελείται κι αυτή από το JCRE όταν λαμβάνεται συγκεκριμένο μήνυμα με το AID applet το οποίο είναι επιλεγμένο. Σε διαφορετική περίπτωση το JCRE επιστρέφει μήνυμα λάθους. Χρησιμοποιείται όταν θέλουμε να εγκαταλείψουμε ένα applet ή και για να επιλέξουμε κάποιο άλλο.

Μετά από την αποστολή ενός μηνύματος στην κάρτα και τη διαχείριση του από το JCRE, αυτό μας αποστέλλει το κατάλληλο μήνυμα για να μας ενημερώσει για τυχών λάθος ή επιτυχή εκτέλεση της εντολής. Για παράδειγμα, αν στείλουμε ένα μήνυμα επιλογής στο JCRE ενός applet του οποίου το AID υπάρχει και η επιλογή εκτελεστεί κανονικά θα μας επιστραφεί η απάντηση 90 00 η οποία συμβολίζει την κανονική εκτέλεση κάθε λειτουργίας. Σε αντίθετη περίπτωση, αν για παράδειγμα δεν υπάρχει applet με το συγκεκριμένο AID, μας επιστρέφεται το κατάλληλο μήνυμα λάθους. Ανάλλογα με την κατηγορία στην οποία ανήκει το λάθος θα επιστραφεί η κατάλληλη απάντηση σύμφωνα με το παρακάτω σχήμα. Ακόμα σε κάποια λάθη τα οποία προβλέπει ο προγραμματιστής φροντίζει η απάντηση που αποστέλλει να εναρμονίζεται με το πρότυπο ISO 7816-4.



Σχήμα 6 Μηνύματα απάντησης σύμφωνα με το πρότυπο ISO 7816-4

Οι εκτός κάρτας εφαρμογές

Κάθε project που ασχολείται με την τεχνολογία των έξυπνων καρτών εκτός από το μέρος που περιλαμβάνει το λογισμικό που προορίζεται για την κάρτα, εμπεριέχει κ την κατασκευή ενός λογισμικού, μέσω του οποίου θα πραγματοποιούνται οι αιτήσεις προς την κάρτα. Αυτό, στην περίπτωση που αναπτύσσουμε μια APDU εφαρμογή, συχνά διαιρείται σε τρεις τάξεις προγραμμάτων. Η μία από αυτές είναι μια εφαρμογή υψηλού επιπέδου, η οποία διαχειρίζεται τις κλήσεις και τις απαντήσεις από και προς την κάρτα σε υψηλό επίπεδο και ονομάζεται η εφαρμογή πελάτη (Client ή Back-End application). Η δεύτερη έχει σαν σκοπό να γεφυρώσει το χάσμα ανάμεσα στην εφαρμογή πελάτη και την εφαρμογή της κάρτας, και ονομάζεται πληρεξούσια εφαρμογή (Proxy ή Host application). Στην περίπτωση που η υπό ανάπτυξη εφαρμογή μας υιοθετεί την λογική του RMI μοντέλου, δεν πρέπει να αναπτύξουμε την πληρεξούσια εφαρμογή, καθώς μια RMI εφαρμογή έχει σαν στόχο να αντικατασταθεί αυτή η χαμηλού επιπέδου επικοινωνία, με την κλήση απομακρυσμένων μεθόδων.

Η εφαρμογή πελάτη όπως ήδη αναφέραμε είναι μια υψηλού επιπέδου εφαρμογή και μπορεί να αναπτυχθεί χωρίς να υπάρχει απαραίτητα γνώση της τεχνολογίας των έξυπνων καρτών. Συγκεκριμένα, έχει την ευθύνη της διασύνδεσης με τον χρήστη, τον μερικό έλεγχο των δεδομένων που προέρχονται από τον χρήστη, την κατάλληλη παρουσίαση τους και τον έλεγχο των εξαιρέσεων που προέρχονται από την πληρεξούσια

εφαρμογή, το τελευταίο για APDU εφαρμογές μόνο. Για την καλύτερη και αποτελεσματικότερη διασύνδεση με τον χρήστη συχνά χρησιμοποιείται γραφικό περιβάλλον, ώστε να διευκολύνει την επικοινωνία με αυτόν. Επίσης το πρόγραμμα πελάτη, στην περίπτωση μίας APDU εφαρμογής, είναι αυτό το οποίο επικοινωνεί με τη πληρεξούσια εφαρμογή ώστε να γίνουν οι κατάλληλες αιτήσεις προς την κάρτα και να ληφθούν έπειτα από αυτές οι εκάστοτε απαντήσεις. Από την άλλη μεριά, στην περίπτωση μιας RMI εφαρμογής, η εφαρμογή πελάτη έχει επίσης τα λιγοστά καθήκοντα που είχε, αλλά αναθέτονται επιπλέον σε αυτήν κάποιες μικρές τροποποιήσεις στην αποστολή κ στην λήψη δεδομένων και η απευθείας κλήση των μεθόδων της κάρτας.

Από τους βασικότερους ρόλους σε ένα APDU έργο έχει η πληρεξούσια εφαρμογή. Είναι αυτή η οποία αναλαμβάνει την επικοινωνία ανάμεσα στην εφαρμογή πελάτη και στην εφαρμογή της κάρτας. Μία πληρεξούσια εφαρμογή αντιστοιχεί σε μία μόνο εφαρμογή της κάρτας και συμφωνεί με αυτή στις εντολές περιέχει, που είναι απαραίτητες για την μεταξύ τους επικοινωνία. Με λίγα λόγια γνωρίζει τις ειδικές εντολές χαμηλού επιπέδου που ορίζονται μέσα στην εφαρμογή της κάρτας, τις γενικές εντολές χαμηλού επιπέδου που προορίζονται για το λογισμικό που διαχειρίζεται την κάρτα και είναι ενσωματωμένο σε αυτές, καθώς και τον κώδικα εξαιρέσεων που είναι πιθανό να εγερθούν από αυτά τα λογισμικά. Επιπλέον, σχετικά με την διαχείριση των εξαιρέσεων, αναλαμβάνει και εδώ τον ρόλο του διαμεσολαβητή, καθώς κατανοεί τις εξαιρέσεις που προέρχονται από την κάρτα, οι οποίες δεν έχουν την κλασσική μορφή των εξαιρέσεων αλλά αναπαριστούνται με την μορφή των δυο bytes, και τις μετατρέπει, κατά την ευθύνη του προγραμματιστή, σε εξαιρέσεις χρόνου εκτέλεσης.

Τα πακέτα αυτά μπορούν να γραφτούν σε οποιαδήποτε γλώσσα, αλλά η γλώσσα που ευνοεί την ανάπτυξη τους είναι αναμφισβήτητα η Java, ειδικά εάν το έργο βασίζεται στην RMI λογική. Σημαντικό είναι να τονίσουμε ότι σε μια APDU εφαρμογή μπορούν να υπάρχουν παραπάνω της μίας, διαφορετικές εφαρμογές πελάτη, αλλά σχεδόν πάντα υπάρχει μόνο μία πληρεξούσια εφαρμογή.

Επικοινωνία εφαρμογών

Σε αυτήν την ενότητα θα αναλυθεί ο τρόπος με τον οποίον γίνεται η ανταλλαγή των πληροφοριών των εφαρμογών που τρέχουν μέσα στην κάρτα και τον εφαρμογών εκτός κάρτας. Επειδή ο τρόπος αυτός εξαρτάται από την τεχνολογία υλοποίησης θα παρουσιαστούν, αρχικά, οι βασικές διαφορές τους και τα επιμέρους στάδια που είναι απαραίτητα για την ολοκλήρωση της επικοινωνίας.

Διαφορές APDU - RMI

Κατά την ανάπτυξη μιας java card θα πρέπει να αποφασιστεί ο εξοπλισμός που θα χρησιμοποιηθεί, κάρτες και readers, ο οποίος επιβάλει και την τεχνολογία υλοποίησης. Αυτό γιατί APDU υποστηρίζουν όλες οι έξυπνες κάρτες ενώ RMI υποστηρίζεται από την έκδοση 2.2 της java card και μετά. Παρακάτω παρουσιάζονται οι βασικότερες διαφορές των δύο τεχνολογιών. Ορισμένες από αυτές θα εξηγηθούν παρακάτω, στην επικοινωνία εφαρμογών.

Πίνακας 2 Διαφορές APDU - RMI

APDU	RMI
Πλήρης έλεγχος των APDU μηνυμάτων	Απόκρυψη των APDU μηνυμάτων
Έκδοση/Δημιουργία εντολών	Κλήση μεθόδων
Χρήση πληρεξούσιας εφαρμογής (proxy)	Χρήση εφαρμογής στελέχους (stub)
Απλότητα Applet εφαρμογής	Πολύπλοκη εφαρμογή Applet
Επικοινωνία των Applet μέσω διαμοιραζόμενης διασύνδεσης	Αδύνατη η υλοποίηση διαμοιραζόμενης διασύνδεσης

Λόγω του διαθέσιμου εξοπλισμού, και συγκεκριμένα λόγο των δεδομένων καρτών, δεν υποστηρίζουν την έκδοση 2.2 της java card, δεν υπάρχει η δυνατότητα ανάπτυξης μιας RMI εφαρμογής σε αυτές. Εξαιτίας αυτών των περιορισμών αναπτύχθηκαν δύο εκδοχές του project, μία παραδοσιακή και μία RMI. Στην πρώτη υπάρχουν δύο παραδοσιακές εφαρμογές ενώ στη δεύτερη μια παραδοσιακή και μια RMI.. Και οι δύο είναι ικανές να τρέχουν σε περιβάλλον προσομοίωσης, αλλά μόνο η πρώτη έχει τη δυνατότητα να εκτελείται σε πραγματικό περιβάλλον.

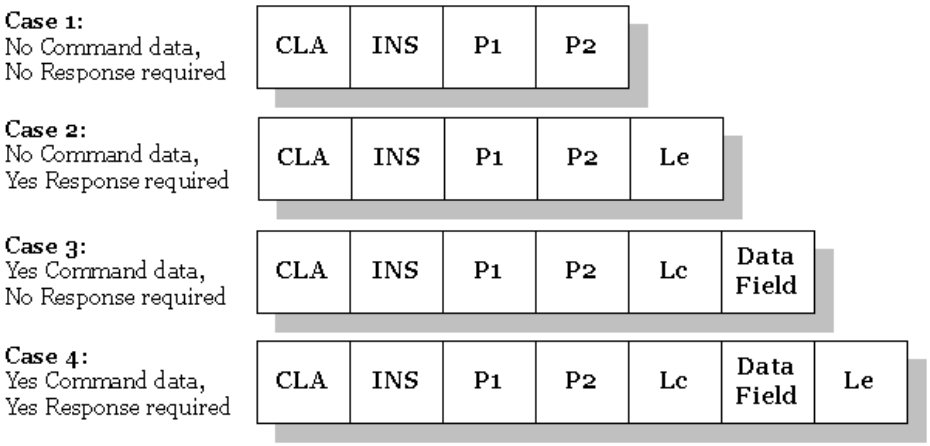
Η επικοινωνία μέσω APDU

Βασικό στοιχείο της τεχνολογίας των έξυπνων καρτών, πάνω στο οποίο βασίζεται όλη η επικοινωνία μεταξύ των της τερματικής συσκευής και της κάρτας, είναι τα πακέτα μηνυμάτων APDU's (Application Protocol Data Unit). Η επικοινωνία κάθε εφαρμογής σε java card, είτε αυτή πρόκειται για κλασσικό APDU μοντέλο είτε για RMI, βασίζεται πάνω στη χρήση αυτών των πακέτων. Η κύρια διαφορά μεταξύ ενός RMI και ενός παραδοσιακού μοντέλου, είναι ότι το πρώτο έχει τη λογική της απόκρυψης αυτών των πακέτων από τον προγραμματιστή, ενώ στο δεύτερο υπάρχει απευθείας εργασία με αυτές τις χαμηλού επιπέδου εντολές.

Τα πακέτα APDU, συμμορφώνονται με τα πρότυπα ISO/IEC 7816-3 και 7816-4 και αποτελούν ένα γενικό πρότυπο επικοινωνίας για συσκευές με περιορισμένη μνήμη. Δεν αναφέρεται μόνο σε κάρτες που υποστηρίζουν java card αλλά γενικότερα σε smart cards. Ένα πακέτο APDU μπορεί να είναι είτε εντολή, είτε απάντηση.

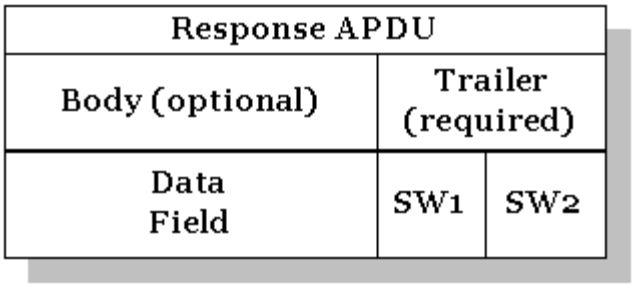
Μια εντολή APDU χωρίζεται σε δύο βασικά πεδία, την κεφαλίδα (header) η οποία είναι υποχρεωτική και το σώμα (body) το οποίο είναι προαιρετικό. Η κεφαλίδα χωρίζεται σε τέσσερα πεδία τα οποία αναπαριστούν ένα byte το καθένα. Το πρώτο πεδίο αναπαριστά την εντολή κλάσης (CLA), το οποίο δείχνει την κλάση που επιθυμούμαι να αναφερθούμε. Το δεύτερο πεδίο καθορίζει μια συγκεκριμένη εντολή (INS) η οποία σχετίζεται με την κλάση που επιλέχθηκε. Τα υπόλοιπα δύο πεδία λειτουργούν σαν ορίσματα ώστε να συγκεκριμενοποιήσουμε την εργασία που ορίζουν τα αρχικά δύο bytes. Έπειτα ακολουθεί το σώμα το οποίο έχει μεταβλητό μέγεθος, καθώς μπορούν να παραληφθούν πεδία τα οποία είναι προαιρετικά όπως και να στείλουμε δεδομένα μεταβλητού μήκους. Το πρώτο στοιχείο αυτού υποδεικνύει το μήκος των δεδομένων που

επιθυμούμαι να στείλουμε (Lc), εκφρασμένο σε ένα byte. Τα επόμενα πεδία που ακολουθούν είναι τα δεδομένα που θα στείλουμε (Data), και έχουν μήκος όσο έχουμε ορίσει στο προηγούμενο πεδίο. Εξαιτίας αυτού του περιορισμού, το ότι ένα μόνο byte περιγράφει τα πόσα δεδομένα θα σταλούν, μπορούμε να στείλουμε μέχρι 255 bytes τη φορά. Το τελευταίο byte περιγράφει το μέγιστο αναμενόμενο μήκος της απάντησης, που θα ληφθεί από την κάρτα. Η εικόνα εμφανίζει όλους τις πιθανές μορφές που μπορεί να έχει μία εντολή APDU.



Σχήμα 7 Πιθανές μορφές μιας εντολής APDU (APDU command)

Αφού στείλουμε την εντολή APDU περιμένουμε να ληφθεί μία απάντηση APDU από την κάρτα. Η δομή αυτού του APDU περιέχει δύο πεδία, το σώμα (Body) το οποίο είναι προαιρετικό και η κατάληξη (Trailer) που είναι υποχρεωτική. Το σώμα περιέχει τα δεδομένα (Data Field) τα οποία πιθανώς να ζητούνται από την εντολή APDU, και η κατάληξη αποτελείται από δύο bytes (SW1,SW2) τα οποία δείχνουν την κατάσταση που έχει επιφέρει η τελευταία εντολή APDU. Το JCRE φροντίζει να μας στέλνει πάντοτε απάντηση, response APDU, για να γνωρίζουμε την έκβαση μιας εντολή, command APDU, που στείλαμε.



Σχήμα 8 Απάντηση APDU (Response APDU)

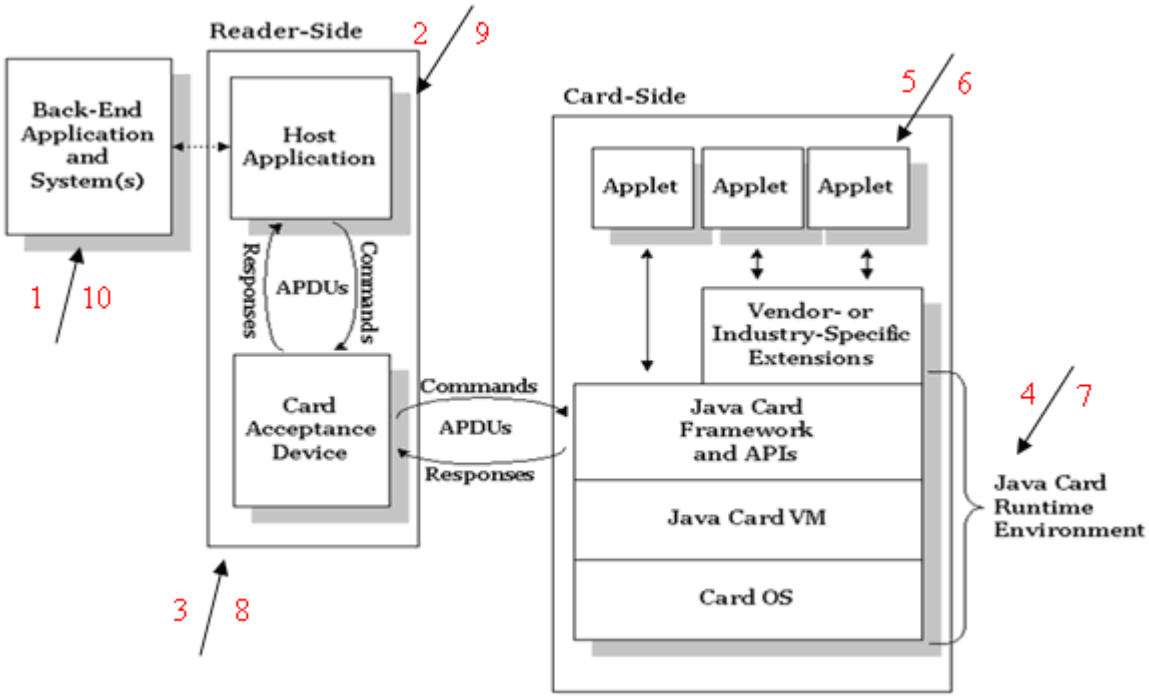
Πολλές φορές ο προγραμματιστής στον κώδικα του applet ορίζει την απάντηση που θα αποστείλει ανάλογα με την εντολή που θα δεχτεί και τις αλλαγές που θα επιφέρει αυτή. Για παράδειγμα, έστω ότι στο applet υποστηρίζεται η λειτουργία αφαίρεσης χρημάτων από το λογαριασμό. Η μέθοδος αυτή δέχεται σαν όρισμα ένα ποσό το οποίο ο χρήστης επιθυμεί να αφαιρέσει, παράδειγμα για την αγορά ενός προϊόντος. Εάν αυτό το ποσό είναι μεγαλύτερο από το υπόλοιπο του λογαριασμού τότε πρέπει να επιστραφεί ένα μήνυμα λάθους. Τα μοναδικά μηνύματα όμως που μπορεί να στείλει η κάρτα είναι τα response APDU. Έτσι αποστέλλουμε τα κατάλληλα bytes τα οποία μας πληροφορούν για το λάθος που προέκυψε.

Οι τιμές των SW1 και SW2, που μας πληροφορούν για το αποτέλεσμα του command APDU, ορίζονται σύμφωνα με το πρότυπο ISO 7816-4. Οι διάφορες τιμές των δύο αυτών bytes περιγράφονται στο σχήμα της παραγράφου Επικοινωνία Applet – Java Card Runtime Environment.

Η επικοινωνία του πελάτη με την κάρτα

Σε κάθε λογισμικού Java Card, το βασικότερο στοιχείο ώστε να επιτελεστεί κάποια διεργασία στην κάρτα, είναι η επικοινωνία αυτής με εξωτερικές εφαρμογές. Για να επιτευχθεί αυτή, σε μία APDU εφαρμογή, χρειάζεται μια αλυσιδωτή επικοινωνία προγραμμάτων, από τα οποία μόνο ένα μέρος είναι ευθύνη του προγραμματιστή. Δηλαδή κάποια μέρη παρέχονται έτοιμα ενώ άλλα πρέπει να υλοποιηθούν. Αφαιρετικά, υπάρχουν τρεις κατηγορίες προγραμμάτων εκτός κάρτας, η Back-End (εφαρμογή πελάτη, client) , η Host (πληρεξούσια εφαρμογή, proxy) και η Card Acceptance Device (CAD). Οι δύο πρώτες είναι αναγκαίο να υλοποιηθούν, ενώ η CAD εφαρμογή παρέχεται ολοκληρωμένη και υλοποιεί την φυσική διασύνδεση μεταξύ της πληρεξούσιας εφαρμογής (Host) και

των εντός κάρτας προγραμμάτων. Η πληρεξούσια και η CAD εφαρμογή συμμορφώνονται με τεχνολογίες ,όπως η OpenCard, το OpenJDK και η satsa, οι οποίες έχουν σαν στόχο την απλοποιημένη επικοινωνία της κάρτας με τις εξωτερικές εφαρμογές. Δηλαδή παρέχουν έτοιμες βιβλιοθήκες για την επικοινωνία της εφαρμογής πελάτη με την κάρτα. Σε ένα RMI έργο όμως, τα πράγματα αλλάζουν. Σε μία τέτοια περίπτωση, οι τρεις παραπάνω κατηγορίες περιπλέκονται με τρόπο ώστε να μην υπάρχει ξεκάθαρη διάκριση των παραπάνω κατηγοριών. Ξεκινώντας από υψηλό επίπεδο, στην περίπτωση APDU τεχνολογίας διακρίνουμε ότι το πρόγραμμα πελάτη επικοινωνεί με την πληρεξούσια εφαρμογή, η οποία στην συνέχεια αναλαμβάνει να στείλει το κατάλληλο μήνυμα στην CAD εφαρμογή, με τελικό προορισμό την κάρτα και συγκεκριμένα σε κάποια εφαρμογή της. Με βάση το παρακάτω σχήμα θα περιγράψουμε ένα σενάριο επικοινωνίας πελάτη – κάρτα. Η διαδικασία ξεκινάει από το σημείο 1, στο σχήμα, και τερματίζει στο 11 και αποτελείται από δύο μέρη. Το πρώτο, αρχίζει από την εφαρμογή πελάτη(σημείο 1) και καταλήγει στην κάρτα (σημείο 5), οι αριθμοί αναγράφονται στα αριστερά του βέλους, παρουσιάζοντας την πορεία μιας εντολής, command APDU, προς την κάρτα και το δεύτερο από το σημείο 6 μέχρι το 11), οι αριθμοί αναγράφονται στα δεξιά του βέλους, όπου γίνεται η αποστολή της απάντησης, response APDU, προς τον πελάτη. Πιο αναλυτικά, όταν ο χρήστης αποφασίσει να εκκινήσει μία διεργασία, μέσω του γραφικού περιβάλλοντος του χρήστη (σημείο 1) εισάγει τα τυχών δεδομένα και έπειτα η εφαρμογή πελάτη τα αποστέλλει ως όρισμα στην πληρεξούσια εφαρμογή (σημείο 2), τις περισσότερες φόρες χωρίς επεξεργασία και σαν αλφαριθμητικό. Στην συνέχεια η πληρεξούσια εφαρμογή, αναλαμβάνει να μετατρέψει τα δεδομένα αυτά σε bytes, και αφού συμπληρώσει τα απαραίτητα πεδία σε μία εντολή APDU, τα τοποθετεί εντός αυτής και αποστέλλει την εντολή στην CAD εφαρμογή (σημείο 3). Η τελευταία, αναλαμβάνει τον έλεγχο της εντολής και την έκφραση της σε λογικά σήματα, όπου με την βοήθεια της συσκευής ανάγνωσης καρτών στέλνει την εντολή στην κάρτα. Έπειτα η κάρτα με τη βοήθεια του Java Card Runtime Environment (σημείο 4) επεξεργάζεται τα δεδομένα και καλεί το κατάλληλο applet (σημείο 5). Αυτό αναγνωρίζει την εντολή και εκτελεί τις κατάλληλες εργασίες (σημείο 6).



Σχήμα 9 Η επικοινωνία προγράμματος πελάτη με την κάρτα

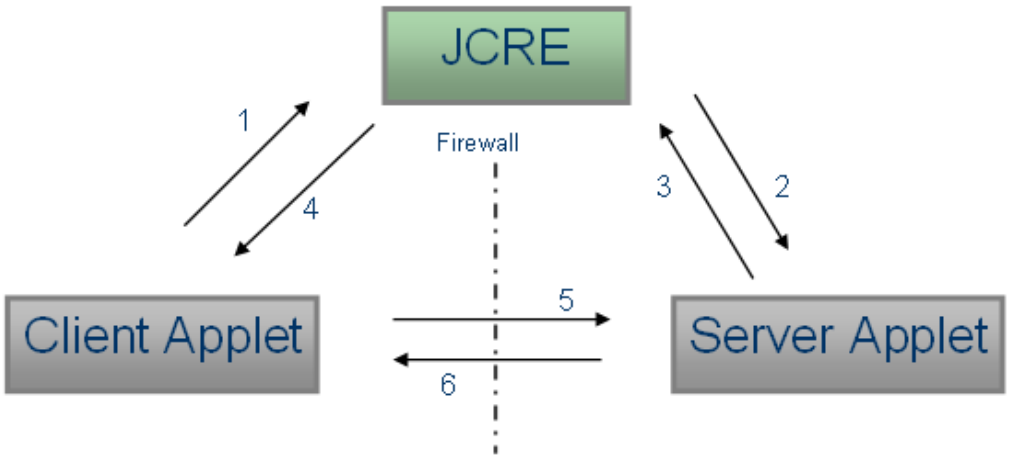
Σε αυτό το σημείο έχει τελειώσει η διαδρομή της εντολής και ξεκινάει η αποστολή της απάντησης, η οποία θα διασχίσει το αντίστροφο μονοπάτι από την εντολή. Έτσι το applet στέλνει, ανάλογα με την πορεία εκτέλεσης της εντολής, την απάντηση στο JCRE (σημείο 7) και αυτό με τη σειρά του πίσω στην CAD εφαρμογή. Η CAD εφαρμογή θα στείλει το μήνυμα της απάντησης στην πληρεξούσια εφαρμογή (σημείο 8), η οποία θα λάβει τα δεδομένα, θα τα μετασχηματίσει και θα τα επιστρέψει σε κατάλληλη μορφή, στον πελάτη (σημείο 9). Τέλος η εφαρμογή πελάτη ενημερώνει το χρήστη (σημείο 10) για τα αποτελέσματα που επέφερε η εντολή που είχε δώσει ο χρήστης στο σημείο 1. Υπάρχει η περίπτωση να παραλειφθούν τα βήματα 5 και 6 αν για παράδειγμα το applet στο οποίο αναφερόμαστε δεν υπάρχει στην κάρτα. Τότε, το JCRE είναι υπεύθυνο για τη δημιουργία και την αποστολή στην CAD εφαρμογή του κατάλληλουresponse APDU.

Η επικοινωνία των εφαρμογών μέσα στην κάρτα

Ένα από τα σημαντικότερα χαρακτηριστικά που έχει παρουσιάσει η τεχνολογία των καρτών, είναι η ελεγχόμενη επικοινωνία εφαρμογών μέσα στην κάρτα. Η μεταξύ τους επικοινωνία ελέγχεται από έναν μηχανισμό του λογισμικού της κάρτας, και ποιο

συγκεκριμένα του Java Card Runtime Environment, ο οποίος ονομάζεται firewall. Οι εφαρμογές στην κάρτα τοποθετούνται μέσα σε πακέτα, τα οποία οριοθετούν την εμβέλεια επικοινωνίας μεταξύ τους. Δύο εφαρμογές που βρίσκονται στο ίδιο πακέτο επικοινωνούν χωρίς να τους επηρεάζει το firewall. Η ευθύνη του firewall είναι να εμποδίσει την επικοινωνία των προγραμμάτων που ανήκουν σε διαφορετικά πακέτα, τα οποία στην πλειονότητα των περιπτώσεων προέρχονται από διαφορετικό κατασκευαστή ή επιχείρηση. Αυτό γίνεται ώστε να μην επιτραπεί η επικοινωνία μεταξύ δυο εφαρμογών, καθώς η μία από αυτές μπορεί να είναι κακόβουλη και μη εξουσιοδοτημένη, με σκοπό να έχει πρόσβαση σε ευαίσθητα δεδομένα. Το firewall όμως μπορεί να παρακαμφτεί με κάποιο τρόπο, ο οποίος ονομάζεται η τεχνική της διαμοιραζόμενης διασύνδεσης (Shareable Interface). Αυτό το χαρακτηριστικό επιτρέπει σε μια εφαρμογή της κάρτας να δηλώσει ποια στοιχεία είναι διατεθειμένη να διαμοιράσει με άλλα προγράμματα, πάντοτε μέσα στην κάρτα. Τις περισσότερες φορές από σχεδιαστικής άποψης τα στοιχεία που διαμοιράζονται είναι μέθοδοι, καθώς επιθυμούμε μια ελεγχόμενη πρόσβαση στις μεταβλητές της κλάσης.

Με βάση το παρακάτω σχήμα θα εξηγήσουμε τα βήματα που εκτελούνται για την ανταλλαγή των δεδομένων. Αρχικά, για να δηλώσει μία εφαρμογή ποιες μεθόδους προτίθεται να διαμοιράσει, δημιουργείται από τον προγραμματιστή μία διαμοιραζόμενη διασύνδεση, στο server applet. Σε αυτή, δηλώνονται οι διαμοιραζόμενες διαδικασίες χωρίς να καθορίζεται η διεργασία που θα επιτελέσουν. Έπειτα η εφαρμογή υλοποιεί αυτή τη διασύνδεση και συνεπώς όλες τις μεθόδους που έχουν οριστεί σε αυτήν, και πρόκειται να διαμοιράσει. Από την άλλη το πρόγραμμα που επιθυμεί να χρησιμοποιήσει τις διαμοιραζόμενες μεθόδους μιας άλλης εφαρμογής, το client applet, κάνει αίτηση στο Java Card Runtime Environment, δίνοντας σαν όρισμα το AID της εφαρμογής που θέλει να επικοινωνήσει (βήμα 1). Στη συνέχεια, το Java Card Runtime Environment επικοινωνεί με την εφαρμογή που έχει το συγκεκριμένο AID και ζητάει από αυτήν, έπειτα από μία προαιρετική πιστοποίηση, την έγκριση για τη διαμοίραση των στοιχείων της.



Σχήμα 10 Διαμοιραζόμενη Διασύνδεση (Sharable Interface)

1 - Αίτηση διαμοιραζόμενης διασύνδεσης από το JCRE
2 - Αίτηση διαμοιραζόμενης διασύνδεσης από το Applet
3 - Επιστροφή διαμοιραζόμενης διασύνδεσης από το Applet
4 - Επιστροφή διασύνδεσης από το JCRE
5 - Χρήση της διασύνδεσης
6 - Επιστροφή αποτελεσμάτων

Με βάση το παραπάνω σχήμα θα εξηγήσουμε τα βήματα που εκτελούνται για την ανταλλαγή των δεδομένων. Αρχικά, για να δηλώσει μία εφαρμογή ποιες μεθόδους προτίθεται να διαμοιράσει, δημιουργείται από τον προγραμματιστή μία διαμοιραζόμενη διασύνδεση, στο server applet. Σε αυτή, δηλώνονται οι διαμοιραζόμενες διαδικασίες χωρίς να καθορίζεται η διεργασία που θα επιτελέσουν. Έπειτα η εφαρμογή υλοποιεί αυτή τη διασύνδεση και συνεπώς όλες τις μεθόδους που έχουν οριστεί σε αυτήν, και πρόκειται να διαμοιράσει. Από την άλλη το πρόγραμμα που επιθυμεί να χρησιμοποιήσει τις διαμοιραζόμενες μεθόδους μιας άλλης εφαρμογής, το client applet, κάνει αίτηση στο Java Card Runtime Environment, δίνοντας σαν όρισμα το AID της εφαρμογής που θέλει να επικοινωνήσει (βήμα 1). Στη συνέχεια, το Java Card Runtime Environment επικοινωνεί με την εφαρμογή που έχει το συγκεκριμένο AID και ζητάει από αυτήν, έπειτα από μία προαιρετική πιστοποίηση, την έγκριση για τη διαμοίραση των στοιχείων της (βήμα 2). Εάν τελικά δεν προκύψει κάποιο σφάλμα, επιστρέφεται στην εφαρμογή που έκανε την αίτηση, ένα στιγμιότυπο της διαμοιραζόμενης διασύνδεσης μέσω του

οποίου μπορεί να χρησιμοποιήσει απευθείας μεθόδους του άλλου προγράμματος (βήματα 3 και 4). Στη συνέχεια το client applet χρησιμοποιεί τη διασύνδεση για να ζητήσει τα δεδομένα που χρειάζεται (βήμα 5) και τέλος το server applet επιστρέφει τα αποτελέσματα στο applet που τα ζήτησε (βήμα 6).

Κατά τη διαδικασία παραγωγής του κώδικα και μετατροπής του σε κώδικα κάρτας είναι απαραίτητο από τη μεριά της εφαρμογής η οποία θα χρησιμοποιήσει τη διαμοιραζόμενη διασύνδεση κάποιας άλλης, να έχει γνώση της διαμοιραζόμενης διασύνδεσης αυτής όπως επίσης και ενός αρχείου το οποίο είναι παράγωγο της διαδικασίας παραγωγής.

Με αυτή η τεχνική μια APDU εφαρμογή μπορεί να διαμοιράσει τα στοιχεία της εύκολα και με ασφάλεια. Αντίθετα, μία RMI εφαρμογή δεν είναι ικανή, λόγο της δομής της, να αξιοποιήσει αυτό το χαρακτηριστικό. Βέβαια μια RMI εφαρμογή μπορεί να χρησιμοποιήσει τη διαμοιραζόμενη διασύνδεση μιας APDU εφαρμογής, σε ένα υβριδικό μοντέλο.

Ανάπτυξη Java Card Εφαρμογών

Σε αυτήν την ενότητα θα ασχοληθούμε με τα μέσα που απαιτούνται για την ανάπτυξη Java Card εφαρμογών καθώς και τη διαδικασία που πρέπει να ακολουθήσει κανείς για την παραγωγή αρχείων ικανών να φορτωθούν στην κάρτα.. Τα μέσα αυτά είναι τα πακέτα λογισμικού και ο απαραίτητος εξοπλισμός όπως οι κάρτες και η Card Acceptance Device. Ακόμα θα αναφερθούν διάφοροι κανόνες συμβατότητας ανάμεσα στα εργαλεία ανάπτυξης. Τέλος, παρουσιάζονται τα βήματα εκτέλεσης της εφαρμογής σε περιβάλλον προσομοίωσης, που παρέχεται από τη Sun, και στο πραγματικό περιβάλλον της κάρτα.

Εργαλεία ανάπτυξης

Για την ανάπτυξη ενός java card project απαιτούνται ορισμένα εργαλεία.. Ένα πακέτο software το οποίο παρέχει εξειδικευμένες βιβλιοθήκες και εργαλεία για την ανάπτυξη και τον έλεγχο των εφαρμογών κάρτας, ένα IDE το οποίο με προσθήκη των προηγούμενων βιβλιοθηκών βοηθά στην ανάπτυξη του κώδικα της εφαρμογής client

αλλά και του applet της κάρτας σε γραφικό περιβάλλον και τέλος ένα πακέτο λογισμικού για τη διαχείριση της κάρτας (Card Manager). Για την ανάπτυξη αυτού του project χρησιμοποιήθηκαν αντίστοιχα το java card development kit 2.2.1, το JBuilder Enterprise 2005 και το JCard Manager 4. Αυτά τα πακέτα λογισμικού περιέχουν επίσης εργαλεία τόσο για εργασία σε περιβάλλον προσομοίωσης, όσο και σε πραγματικό περιβάλλον.

Αν και το JCDK 2.2.1 δεν είναι το πιο νέο περιβάλλον που έχει εκδώσει η sun, αυτή τη στιγμή κυκλοφορεί η έκδοση 2.2.2. και ετοιμάζεται η έκδοση 3, είναι αναγκαία η ανάπτυξη των εφαρμογών με αυτό. Τα τελικά αρχεία, όμως, πρέπει να είναι συμβατά με την έκδοση 2.1.1. η οποία υποστηρίζεται από τις δεδομένες κάρτες, το οποίο επιτυγχάνεται με την προσθήκη ορισμένων αρχείων της έκδοσης αυτής στο τρέχων περιβάλλον. Ακόμα ένας λόγος που εργαζόμαστε με το JCDK 2.2.1 είναι ότι παρέχει πιο σύγχρονα εργαλεία ανάπτυξης.

Integrated Development Environment

Η πλειοψηφία των λογισμικών που αναπτύσσονται στις μέρες μας βασίζονται σε κάποιο Integrated Development Environment (I.D.E.), λόγο των ευκολιών που παρέχουν αυτά τα περιβάλλοντα. Η υλοποίηση ενός java card project μπορεί να επιτευχθεί σε ένα τέτοιο περιβάλλον καθώς είναι ένα υποσύνολο της java. Υπάρχει η δυνατότητα ανάπτυξης και των δύο μερών του προγράμματος στο ίδιο project, αλλά και σε ξεχωριστά projects. Αυτό εξαρτάται από τη μέθοδο που θα επιλέξουμε για να δομήσουμε την εφαρμογή της κάρτας, RMI ή παραδοσιακή. Απαραίτητο βήμα για την ρύθμιση των παραμέτρων του περιβάλλοντος είναι η εισαγωγή ορισμένων απαραίτητων βιβλιοθηκών για την εκτέλεση των προγραμμάτων. Πρέπει να δοθεί προσοχή σε αυτό καθώς πρέπει να εισαχθούν μόνο οι απαραίτητες βιβλιοθήκες καθώς οι επιπρόσθετες υπάρχει μεγάλη πιθανότητα να δημιουργήσουν πρόβλημα. Κάποιες απαραίτητες είναι αυτές που παρέχει το JCDK και ανάλογα αν πρόκειται για προσομοίωση ή για πραγματική εκτέλεση, βιβλιοθήκες που παρέχει πάλι το JCDK ή η GEMPLUS αντίστοιχα. Επίσης ένα βασικό πλεονέκτημα είναι η υποστήριξη της εύκολης δημιουργίας προγραμμάτων με γραφικό περιβάλλον. Στο συγκεκριμένο project, προτιμήθηκε η συγγραφή του κώδικα να πραγματοποιείται στο JBuilder, ενώ η μεταγλώττιση του μέσω της γραμμής εντολών του

DOS. Έτσι, είναι ικανή η παραγωγή των αρχείων κάρτας από το source κώδικα «με το πάτημα ενός κουμπιού».

Java Card Development Kit

Το jcdk, όπως προαναφέρθηκε, είναι ένα πακέτο λογισμικού που διατίθεται δωρεάν από την Sun και παρέχει βιβλιοθήκες που ορίζουν το σύνολο της java card και εργαλεία τόσο για την ανάπτυξη όσο και για τον έλεγχο των εφαρμογών της κάρτας. Κάθε εφαρμογή κάρτας χρησιμοποιεί μόνο τα συγκεκριμένα APIs καθώς αυτές οι βιβλιοθήκες είναι κατασκευασμένες για ανάπτυξη προγραμμάτων που τρέχουν σε συσκευές με περιορισμένη μνήμη. Περιέχουν από βασικές κλάσεις για το χειρισμό βασικών στοιχείων μέχρι και επεκτάσεις κρυπτογραφίας. Όλα τα εργαλεία του JCDK τρέχουν μέσω της γραμμής εντολών, για τα Windows. Ενδεικτικά, παρέχει εργαλεία για παραγωγή αρχείων κάρτας, για τον έλεγχο τους, καθώς και βοηθητικά προγράμματα για την προσομοίωση. Δηλαδή παρέχονται όλες οι βιβλιοθήκες και τα εργαλεία που είναι απαραίτητα για την ανάπτυξη μιας Java Card εφαρμογής. Η εφαρμογή που αναπτύχθηκε χρησιμοποιεί την έκδοση 2.1.1 του jcdk.

Card Manager

Κατηγορία λογισμικού μέσω του οποίου διαχειριζόμαστε την κάρτα. Είναι απαραίτητο για την εργασία μας σε πραγματικό περιβάλλον, δηλαδή σε κάρτες . Μέσω αυτού έχουμε την δυνατότητα να εισάγουμε τα προγράμματα μας στην κάρτα και να τα διαχειριστούμε όπως επιθυμούμε. Πιο αναλυτικά μπορούμε να:

- φορτώσουμε τα προγράμματα μας στην κάρτα
- τα διαγράψουμε
- τα εκτελέσουμε στέλνοντας απ' ευθείας στην κάρτα εντολές χαμηλού επιπέδου (APDU commands) και λαμβάνοντας απαντήσεις
- στείλουμε κρυπτογραφημένες εντολές
- ορίσουμε έναν προσωπικό αριθμό αναγνώρισης (PIN)
- εισάγουμε και να λάβουμε δεδομένα
- διαχειριστούμε την κατάσταση της κάρτας
- εγγράψουμε και να εκτελέσουμε αρχείο σειράς εντολών (script)

Σε περίπτωση που δεν επιθυμούμε τη φόρτωση των εφαρμογών κάρτας σε πραγματικές κάρτες, αλλά να εργαστούμε σε περιβάλλον προσομοίωσης, δε μας χρησιμεύει το λογισμικό αυτό. Ακόμα, απαιτείται ιδιαίτερη προσοχή κατά την επιλογή του συγκεκριμένου Card Manager με το οποίο θα εργαστούμε ώστε αυτό να είναι συμβατό τόσο με τις κάρτες μας όσο και με την έκδοση του Java Card Development Kit (JCDK) με το οποίο αναπτύχθηκε η εφαρμογή. Για αυτό το λόγο, μαζί με την αγορά μιας έξυπνης κάρτας δίνεται και λογισμικό συμβατό με αυτές.. Γνωστά λογισμικά της κατηγορίας αυτή είναι το open source GPShell αλλά και το JCard Manager της Gemplus. το οποίο χρησιμοποιήθηκε για την αποστολή της εφαρμογής που αναπτύξαμε στην κάρτα.

Το λογισμικό αυτό, συνεργάζεται με κάρτες οι οποίες πληρούν τις προδιαγραφές Java Card 2.2.1. – Global Platform 2.1.1. και τις Java Card 2.1.1. – Global Platform 2.0.1.. Οι κάρτες που χρησιμοποιήθηκαν υλοποιούν τις προδιαγραφές Java Card 2.1.1. – Global Platform 2.0.1.. Τέλος, είναι σημαντικό να τονίσουμε ότι ο Jcard Manager δεν προσφέρει εργαλεία ανάπτυξης, αλλά εστιάζει στην διαχείριση της κάρτας.

Εκτός από το λογισμικό Jcard Manager, το πακέτο της Gemplus περιέχε και δύο ειδών κάρτες, τις GemXpresso και GemSafe Xpresso. Οι πρώτες έχουν μέγεθος μνήμης 64K ενώ οι δεύτερες έχουν χωρητικότητα 32K. Αν και φαινομενικά οι δύο κάρτες προορίζονται για την ίδια εργασία, στην πραγματικότητα η κάθε μία έχει διαφορετικό πεδίο χρήσης.

Οι GemXpresso έχουν κατασκευαστεί με σκοπό να μπορούν να υποστηρίξουν προγράμματα που συμμορφώνονται με τις προδιαγραφές Java Card 2.1.1. – Global Platform 2.0.1.. Οι κάρτες αυτές περιέχουν προεγκατεστημένα πακέτα λογισμικού, όπως Card Manager, ή το Java Card Runtime Environment και είναι έτοιμα να αλληλεπιδράσουν με το λογισμικό το οποίο θα εισαχθεί αργότερα στην κάρτα. Η ανάπτυξη των εφαρμογών που προορίζονται για την κάρτα είναι πάντα ευθύνη του κατασκευαστή λογισμικού και επιτελούν την εργασία που αυτός θα ορίσει. Η διαχείριση αυτών των καρτών μπορεί να επιτευχθεί μέσω του Jcard Manager.

Αντίθετα με τις GemXpresso, οι GemSafe Xpresso δεν είναι κάρτες οι οποίες προορίζονται για αποθήκευση εφαρμογών μετά από την έκδοση της. Πιο συγκεκριμένα οι εν λόγο κάρτες περιέχουν μία εφαρμογή, την GemSafe. Είναι δηλαδή έτοιμες και

προορισμένες για χρήση, αλλά όχι για ανάπτυξη. Η GemSafe εφαρμογή επιτρέπει την αποθήκευση ψηφιακών πιστοποιητικών, ψηφιακών υπογραφών, κλειδιών κρυπτογραφίας αλλά μπορεί να χρησιμοποιηθεί και σαν ηλεκτρονικό «κλειδί», όπως για παράδειγμα για την πρόσβαση σε κάποιο υπολογιστικό σύστημα. Η διαχείριση της πραγματοποιείται μέσω της εφαρμογής GemSafe Toolbox. Υπάρχει όμως η δυνατότητα εκμετάλλευσης της μέσω των φυλομετρητών ιστοσελίδων (Web Browser) ή του ηλεκτρονικού ταχυδρομείου, για την ασφαλή χρήση κρυπτογραφικών δεδομένων μέσω του διαδικτύου, αλλά και σε ξένα υπολογιστικά συστήματα.

Κύκλος Ανάπτυξης

Ο κύκλος ανάπτυξης του λογισμικού της κάρτας περιλαμβάνει ορισμένα υποχρεωτικά στάδια όπως η ανάπτυξη του κώδικα, η μετατροπή αυτού σε μορφή ικανή για την μεταφορά του στην κάρτα και κάποια προαιρετικά όπως ο έλεγχος εγκυρότητας (validation) του τελικού αρχείου και η προσομοίωση της εφαρμογής.

Αρχικά πρέπει να γραφτεί ο κώδικας, ύστερα να γίνει αποσφαλμάτωση του και να περάσουμε στο στάδιο της μεταγλώττισης όπου θα παραχθούν τα μεταγλωττισμένα αρχεία. Αυτές οι εργασίες μπορούν να γίνουν σε κάποιο IDE, στην συγκεκριμένη περίπτωση είναι ο JBuilder, ώστε να υπάρχει καλύτερη διαχείριση του κώδικα. Μέχρι στιγμής η διαδικασία είναι ακριβώς η ίδια με αυτή που ακολουθείται στην ανάπτυξη κάθε εφαρμογής. Τα επόμενα βήματα διαφοροποιούνται, καθώς πρέπει να γίνει μετατροπή των αρχείων σε αρχεία κάρτας και η φόρτωση τους σε κάποιο προσομοιωτή ή σε κάποια κάρτα.

Ανάπτυξη του κώδικα

Όπως προαναφέρθηκε για την ανάπτυξη του κώδικα χρησιμοποιήθηκε το IDE JBuilder. Σε αυτόν μπορεί να αναπτυχθεί ο κώδικας της κάρτας όσο και ο κώδικας της host εφαρμογής. Παρόλα αυτά, είναι δυνατή η εκτέλεση μόνο της πληρεξούσιας εφαρμογής και της εφαρμογής πελάτη. Η εφαρμογή της κάρτας μπορεί μόνο να μεταγλωττιστεί καθώς για την εκτέλεση της απαιτούνται κλήση άλλων διεργασιών αφού παραχθούν τα μεταγλωττισμένα αρχεία. Κατά τη διάρκεια ανάπτυξης των δύο εκτός κάρτας εφαρμογών, στην περίπτωση μιας APDU εφαρμογής, δεν είναι αναγκαία η παρουσία του κώδικα της κάρτας στο project του JBuilder. Το αντίθετο συμβαίνει κατά

την παραγωγή μιας RMI εφαρμογής. Τα αρχεία που προορίζονται για την κάρτα πρέπει να είναι παρόν καθώς η εκτός κάρτας εφαρμογή πρέπει να γνωρίζει Κατά τη μεταγλώττιση του κώδικα της κάρτας πρέπει να δοθεί προσοχή σε ορισμένες αυτοματοποιημένες διεργασίες του JBuilder οι οποίες δημιουργούν προβλήματα στον κώδικα.

Βασικό στοιχείο για την ανάπτυξη της εφαρμογής είναι η εισαγωγή ορισμένων, αλλά όχι όλων, βιβλιοθηκών που παρέχονται από το java card development kit. Είναι απαραίτητη η εισαγωγή τους καθώς αυτές είναι τα API's της java card.

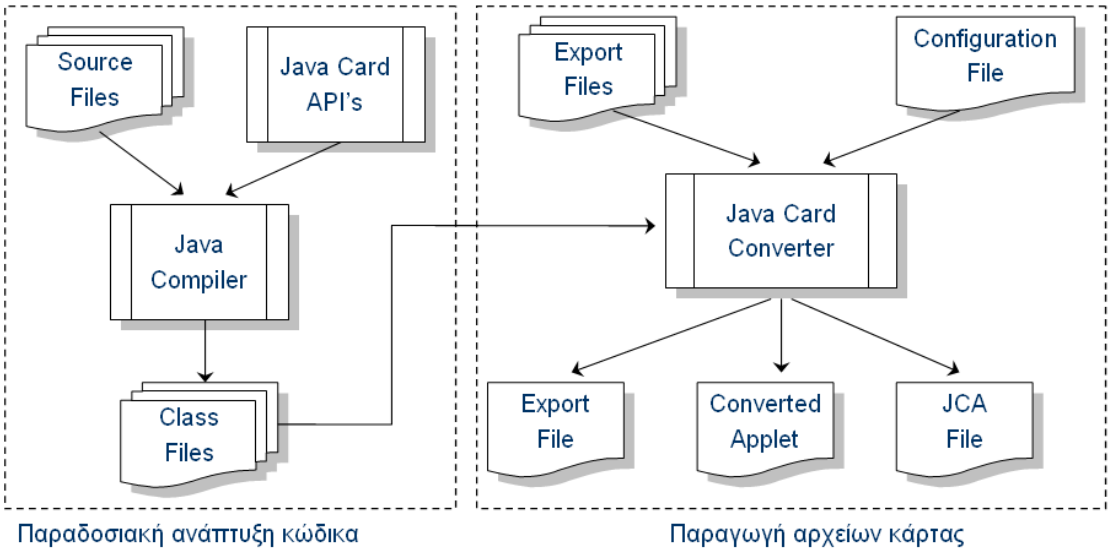
Παραγωγή αρχείων κάρτας

Αφού γίνουν όλα τα προηγούμενα βήματα και παραχθούν τα τελικά class αρχεία, γίνεται η μετατροπή τους σε ένα αρχείο .cap, με το εργαλείο converter που μας παρέχει το JCDK, το οποίο είναι έτοιμο να μεταφερθεί στην κάρτα. Σε κάθε .cap αρχείο μπορούμε να έχουμε ένα ή περισσότερα packages. Ο όρος packages ταυτίζεται με την έννοια packages όπως ορίζεται στην java. Αυτό το βήμα είναι υποχρεωτικό καθώς οι κάρτες είναι έτοιμες να δεχθούν αρχεία μόνο αυτής της μορφής. Αυτά τα αρχεία περιέχουν πακέτα λογισμικού που έχουν μετατραπεί σε κώδικα ικανό να εκτελεστεί στο περιβάλλον της κάρτας. Έτσι και αλλιώς το ακρώνυμο cap σημαίνει Converted Applet. Η τελευταία διαδικασία μάλιστα παράγει και άλλου δύο τύπου αρχεία τα Export (.exp) και τα Java Card Assembly (.jca), τα οποία παράγονται προαιρετικά κατόπιν εντολής.

Για να λειτουργήσει ο converter χρειάζεται στην είσοδό του κάποια ορίσματα από τα οποία τα σημαντικότερα που πρέπει να διοχετευτούν είναι τα export files, το πακέτο που προορίζουμε για μετατροπή, τον αναγνωριστικό αριθμό της έκδοσης του πακέτου και του applet (AID), και την έκδοση του πακέτου αυτού.

Τα export files είναι αρχεία απαραίτητα για τη μετατροπή καθώς περιέχουν πληροφορίες για τα αντικείμενα ή τις βιβλιοθήκες με τα οποία αλληλεπιδρά το υποψήφιο προς μετατροπή applet. Σε κάθε μετατροπή υποχρεωτικά πρέπει να συμπεριληφθούν τα export files της έκδοσης του JCDK, με την οποία θέλουμε να είναι συμβατό το applet, τα οποία δίνουν πληροφορίες στον converter για τις βιβλιοθήκες της συγκεκριμένης έκδοσης, χωρίς φυσικά να τις αντικαθιστούν. Υπάρχει δηλαδή η δυνατότητα να διοχετεύσουμε στον converter μίας συγκεκριμένης έκδοσης της java card, export files

προηγούμενης έκδοσης, ώστε να παράγουμε cap αρχεία αυτής της έκδοσης. Επίσης αν το applet που μετατρέπουμε θα αλληλεπιδράσει με εφαρμογές που βρίσκονται ήδη μέσα στην προοριζόμενη κάρτα, πρέπει να συμπεριληφθούν τα export files που θα παραχθούν από το conversion αυτών των εφαρμογών. Στο παρακάτω σχήμα φαίνεται η διαδικασία παραγωγής αρχείων κάρτας.



Σχήμα 11 Διαδικασία παραγωγής αρχείων κάρτας, .cap

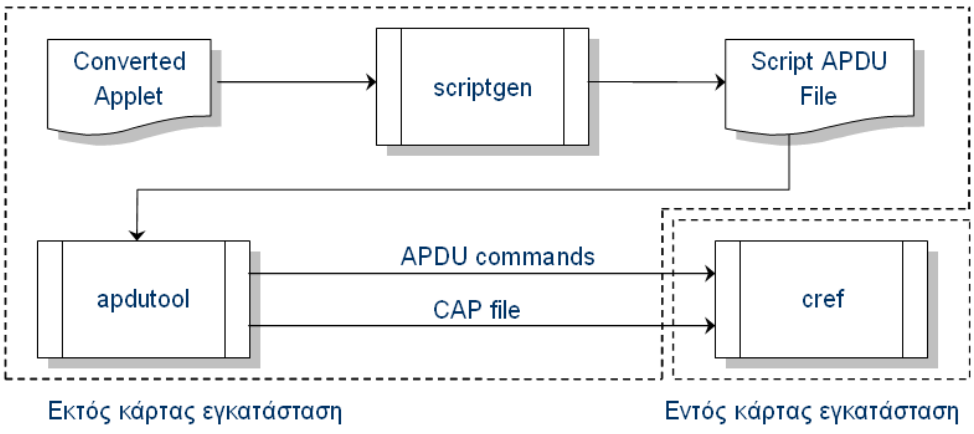
Όλα τα ορίσματα του converter, τα οποία καθορίζουν το converted αρχείο, μπορούν να εισαχθούν μέσω της γραμμής εντολών χειροκίνητα αλλά και να διοχετευτούν από ένα αρχείο (configuration file) που θα δημιουργήσει ο developer, πάλι χειροκίνητα. Στη δεύτερη περίπτωση πρέπει να δημιουργηθεί ένα αρχείο επέκτασης opt το οποίο περιέχει τα ίδια ακριβώς ορίσματα που θα εισαχθούν από τη γραμμή εντολών. Η δημιουργία αυτού του αρχείου είναι προαιρετική, απλά γίνεται για την αυτοματοποίηση της διαδικασίας.

Εκτέλεση της εφαρμογής

Σε αυτό το σημείο, αφού έχουμε ολοκληρώσει την ανάπτυξη της εφαρμογής, υπάρχουν δύο επιλογές. Η μία είναι να ακολουθήσουμε το δρόμο της προσομοίωσης και η άλλη να προχωρήσουμε στη φόρτωση του applet σε μία πραγματική κάρτα. Η λειτουργία της προσομοίωσης παρέχεται από το πακέτο JCDK και υλοποιείται με δύο εργαλεία, το cref και το jcwde, ενώ για να επιτευχθεί η λειτουργία σε πραγματική κάρτα γίνεται χρήση του εργαλείου JCard Manager.

Προσομοίωση

Αν διαλέξουμε τον δρόμο της προσομοίωσης μπορούμε να επιλέξουμε ανάμεσα στο cref και στο jcwde, τα οποία προσομοιώνουν το περιβάλλον της κάρτας μεν, αλλά με κάποιες διαφορές από την πραγματικότητα δε. Το εργαλείο που προσεγγίζει περισσότερο την πραγματικότητα είναι το cref καθώς υποστηρίζει λειτουργίες όπως λογικά κανάλια, εισαγωγή διαγραφή object κ.α.. Η επιτυχία αυτή οφείλεται στην αποθήκευση του συνολικού κώδικα της κάρτας σαν «εικονικό» αρχείο μιας EEPROM η οποία ανά πάσα στιγμή μπορεί να διαβαστεί ή/και να τροποποιηθεί. Η διαδικασία δημιουργίας μίας τέτοιας «εικονικής» EEPROM μοιάζει αρκετά με την αποθήκευση μίας τέτοιας εφαρμογής σε πραγματική κάρτα.



Σχήμα 12 Προσομοίωση με το εργαλείο cref

Όπως παρατηρούμε και στο σχήμα, αφού δημιουργήσουμε το αρχείο με τον κώδικα κάρτας (.cap) δημιουργούμε με την βοήθεια του scriptgen ένα script αρχείο, το Script APDU, που περιέχει εντολές που θα σταλούν στην κάρτα για την αποθήκευση των δεδομένων αυτών. Όταν κατασκευάσουμε αυτό το αρχείο στέλνουμε τις εντολές που περιέχει στην κάρτα με την βοήθεια του apdutool, αφού προηγουμένως έχουμε εκτελέσει το cref σε λειτουργία ανοίγματος αρχείου για εγγραφή, το οποίο θα είναι και η «εικονική» μας μνήμη.

Από την άλλη μεριά το jcwde δεν δημιουργεί «εικονικά» αρχεία EEPROM αλλά εργάζεται κατευθείαν με τα πακέτα κώδικα που έχουν δημιουργηθεί, class αρχεία. Δηλαδή διοχετεύονται σε αυτό config αρχεία στα οποία ορίζονται τα πακέτα που θα χρησιμοποιηθούν και γίνεται η προσομοίωση. Και στις δύο περιπτώσεις ανοίγεται κάποια θύρα επικοινωνίας, εξ' ορισμού η θύρα 9025, η οποία περιμένει αιτήματα, αλλά και στέλνει απαντήσεις, από και σε κάποια εφαρμογή client.

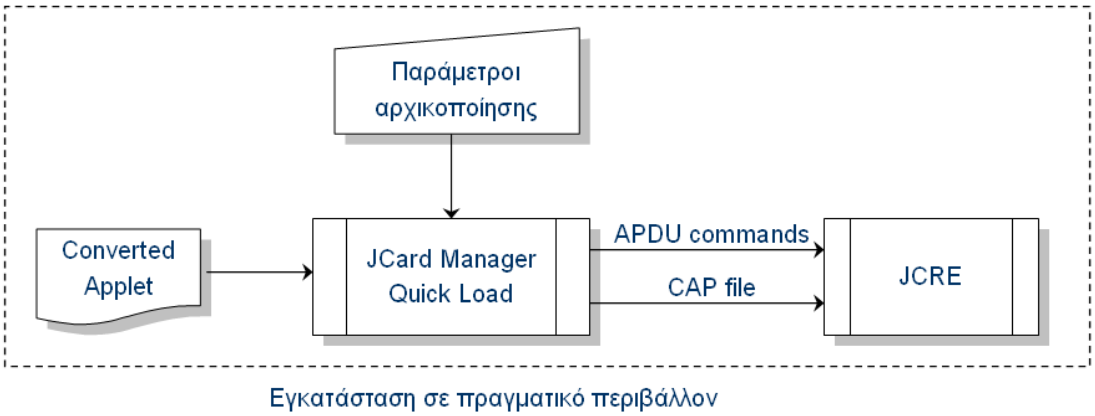
Αφού γράψουμε τον κώδικα σύμφωνα με τα specifications που έχει ορίσει η sun και παράγουμε το αρχείο με τον κώδικα κάρτας, η επόμενη διεργασία, εφόσον θέλουμε να δουλέψουμε σε περιβάλλον προσομοίωσης με το cref, είναι η κατασκευή ενός αρχείου script, επέκτασης .scr, από το αρχείο με τον κώδικα κάρτας. Σε αυτό περιέχονται εντολές οι οποίες θα σταλούν στην εικονική EEPROM, που στην ουσία είναι το cap αρχείο εκφρασμένο σε APDU's. Τα APDU's αυτά είναι πακέτα δεδομένων που χρησιμεύουν για την επικοινωνία της κάρτας με άλλες συσκευές και είναι δεκαεξαδικής μορφής. Η εργασία του αρχείου script είναι η φόρτωση του αρχείου cap στην κάρτα. Αυτό το αρχείο script δημιουργείται από το εργαλείο scriptgen, με είσοδο το αρχείο του κώδικα της κάρτας. Το αρχείο όμως αυτό δεν είναι τελικό καθώς πρέπει να προστεθεί στην αρχή και στο τέλος του κάποιες εντολές, χειροκίνητα.

Αφού διεκπεραιωθούν τα παραπάνω, είμαστε έτοιμοι να δημιουργήσουμε ένα αρχείο εικονικής EEPROM. Αυτό γίνεται εκτελώντας μέσα από την γραμμή εντολών την εφαρμογή cref με όρισμα ανοίγματος αρχείου για αποθήκευση και έπειτα το όνομα του αρχείου εικόνας που θέλουμε να αποθηκεύσουμε τα δεδομένα. Αν το αρχείο αυτό δεν υπάρχει το cref το δημιουργεί, ενώ αν υπάρχει διαγράφονται όλα τα παλιά δεδομένα του και αποθηκεύονται σε αυτό τα νέα. Όταν αρχικοποιηθεί το αρχείο εικόνας έχουμε την δυνατότητα να το ανοίξουμε για ανάγνωση, εκτελώντας πάλι το cref άλλα αυτή την φορά

με επιλογή ανοίγματος για ανάγνωση και το όνομα του αρχείου της εικόνας. Σε αυτήν την περίπτωση, οποιαδήποτε αλλαγή στα δεδομένα της κάρτας πραγματοποιηθεί κατά τη διάρκεια της προσομοίωσης, δεν θα γίνει αποθηκευτεί στο αρχείο εικόνας. Τέλος υπάρχει η δυνατότητα να αποθηκεύσουμε τις μεταβολές των δεδομένων ενός προϋπάρχοντος αρχείου εικόνας οι οποίες θα λάβουν μέρος κατά την διάρκεια ης προσομοίωσης. Αυτό γίνεται απλά ανοίγοντας το αρχείο που περιέχει μια αρχικοποιημένη εικόνα με επιλογή ανάγνωσης και επιπροσθέτως ανοίγοντας, στην ίδια εντολή, ένα άλλο αρχείο (ή και το ίδιο) για εγγραφή των νέων δεδομένων, πάντα με το cref.

Πραγματικό Περιβάλλον

Για να εισάγουμε στην κάρτα μία εφαρμογή απαιτείται η χρήση ενός ειδικού πακέτου λογισμικού που να υποστηρίζει το JCDK σύμφωνα με το οποίο αναπτύχθηκε η εφαρμογή. Στο project μας, χρησιμοποιούμε το λογισμικό JCardManager 4.0 το οποίο επιτρέπει τη συνεργασία με αρχεία κάρτας της έκδοσης JCDK 2.1.1. σύμφωνα με την οποία τα αρχεία που θα σταλούν στην κάρτα πρέπει να έχουν κατάληξη .cap είτε .jar. Μέσω της επιλογή Quick Load από τον JCardManager πραγματοποιείται η φόρτωση αρχείων στην κάρτα, ενώ στο παράθυρο μηνυμάτων του JCardManager παρακολουθούμε την αποστολή των APDUs που διεκπεραιώνουν την παραπάνω διαδικασία καθώς και τις απαντήσεις που αποστέλλονται από την μεριά της κάρτας. Έτσι, κατά τη φόρτωση του κώδικα στην κάρτα αποστέλλουμε την κατάλληλη εντολή APDU μαζί με τον .cap αρχείο. Ταυτόχρονα πρέπει να σταλούν και κάποιες παράμετροι αρχικοποίησης όπως το AID της εφαρμογής, δεδομένα προς αρχικοποίηση αλλά και το κλειδί πιστοποίησης. Η διαδικασία της φόρτωσης φαίνεται στο παρακάτω σχήμα.



Σχήμα 13 Διαδικασία εγκατάστασης του κώδικα στην κάρτα

Ακόμα δίνεται η δυνατότητα διαγραφής μιας εφαρμογής (applet) ή ενός πακέτου από την κάρτα με την επιλογή Delete. Απλά ζητείται από το χρήστη να πληκτρολογήσει το AID της εφαρμογής που θέλει να διαγράψει. Στην περίπτωση που επιθυμούμαι να διαγράψουμε κάποιο πακέτο πρέπει αυτό να είναι άδειο, να μην περιέχει κάποια εφαρμογή, έτσι πρώτα διαγράφουμε όλες τις εφαρμογές που περιέχονται σε ένα πακέτο και στη συνέχεια το ίδιο το πακέτο. Επίσης το λογισμικό υποστηρίζει και ένα πλήθος επιλογών όπως είναι η αποστολή APDU (Send APDU), η οποία μας επιτρέπει να παρακολουθούμε την αντίδραση της κάρτας μέσω των απαντήσεων που αποστέλλει καθώς και η Get Status μέσω της οποίας λαμβάνει πληροφορίες για τις εφαρμογές που περιέχει η κάρτα.

Παρουσίαση παραδείγματος

Η γενική ιδέα είναι ότι μία εφαρμογή ηλεκτρονικού πορτοφολιού αλληλεπιδρά με μια άλλη η οποία θα αποθηκεύει μονάδες επιβράβευσης, όπου τα δεδομένα της δεύτερης εξαρτώνται από τις συναλλαγές της πρώτης. Με λίγα λόγια η δεύτερη εφαρμογή αποτελεί τις μονάδες bonus που μπορεί να κερδίσει κάποιος ως επιβράβευση των συναλλαγών που πραγματοποιεί. Επειδή αυτές οι εφαρμογές απαιτούν ένα ιδιαίτερο επίπεδο ασφάλειας, αυτό θα συμβάλλει στην εξερεύνηση της ασφάλειας που παρέχει η τεχνολογία αυτή, τόσο κατά την επικοινωνία με εφαρμογές εκτός κάρτας, όσο και με εφαρμογές που βρίσκονται μέσα στην ίδια κάρτα.

Από της εφαρμογές που αναπτύχθηκαν θα εξηγήσουμε τον τρόπο εγκατάστασης της εφαρμογής προσομοίωσης που αναπτύχθηκε με APDU και βρίσκεται στο:
Κώδικας\APDU εφαρμογές\Εφαρμογή για εκτέλεση με προσομοίωση

Για την παρουσίαση του παραδείγματος πρέπει αρχικά να εξηγήσουμε την διαδικασία της εγκατάστασης του JCDK και να ορίσουμε τους φακέλους που περιλαμβάνουνε τα αρχεία του project μας. Απαραίτητη προϋπόθεση αποτελεί η εγκατάσταση ενός JDK, το οποίο θα είναι συμβατό, όπως ορίζεται από την sun, με το JCDK που επιθυμούμε να εργαστούμε. Στην συγκεκριμένη περίπτωση χρησιμοποιούμε την έκδοση 2-2-1 του JCDK και το συμβατό προς αυτήν JDK, j2sdk1.4.1_06. Όλη η διαδικασία λαμβάνει μέρος στο περιβάλλον των Windows XP SP2 και οι διεργασίες του JCDK μέσω της γραμμής εντολών DOS.

Εγκατάσταση του JCDK

Αφού κατεβάσουμε το αρχείο μορφής zip από το site της sun (http://java.sun.com/products/javacard/), το αποσυμπιέζουμε στο μονοπάτι `C:\`. Συγκεκριμένα στο παράδειγμα μας αφού χρησιμοποιούμε την έκδοση 2-2-1 του JCDK, έχουμε τον φάκελο `C:\java_card_kit-2_2_1` ο οποίος περιέχει όλα τα απαραίτητα συστατικά για την ανάπτυξη ενός java card project και αποτελεί το Home Path του JCDK. Η διαδικασία της εγκατάστασης δεν έχει τελειώσει, καθώς πρέπει να δημιουργήσουμε ένα αρχείο δέσμης όπου θα δημιουργούμε κάποιες μεταβλητές

περιβάλλοντος. Έτσι δημιουργούμε το αρχείο `environment.bat`, το οποίο αποτελείται από τις παρακάτω εντολές:

```
set JAVA_HOME=C:\Java\j2sdk1.4.1_06
set JC_HOME=C:\java_card_kit-2_2_1
set PATH=.;%JC_HOME%\bin;%JAVA_HOME%\bin;%PATH%
set CLASSPATH=%JC_HOME%\lib\api.jar;%CLASSPATH%
```

Σε αυτές είναι φανερό ότι θέτουμε τα home directories των JDK και JCDK (στην πρώτη και στην δεύτερη εντολή αντίστοιχα) και ύστερα κάνουμε γνωστό στο σύστημα τις τοποθεσίες των εργαλείων ανάπτυξης (development tools) τόσο του JCDK, όσο και του JDK. Στην τελευταία εντολή ορίζεται το μονοπάτι των βιβλιοθηκών της java card. Αυτή η εντολή χρειάζεται ιδιαίτερη προσοχή αφού μπορεί να προϋπάρχει στο σύστημα και να είναι ορισμένη από ένα άλλο πρόγραμμα. Στη δικιά μας περίπτωση απλώς διαγράφτηκε η μεταβλητή περιβάλλοντος χωρίς καμία παρενέργεια. Αυτό γίνεται από Έναρξη->Πίνακας Ελέγχου->Σύστημα->στην ταμπέλα Για προχωρημένους->Μεταβλητές Περιβάλλοντος

Κάθε φορά που επιθυμούμε να κάνουμε κάποια διεργασία στο περιβάλλον της java card, αφού ανοίξουμε ένα νέο παράθυρο γραμμής εντολών, τρέχουμε μέσα από αυτό το batch αυτό αρχείο.

Το project

Ο φάκελος που περιέχει όλα τα αρχεία του project, στη δικιά μας περίπτωση, είναι ο `C:\JCard_APDU_sim` και περιέχει όλα τα αρχεία της host εφαρμογής αλλά και της κάρτας. Τα αρχεία της εφαρμογής του client, τα οποία έχουν και γραφική διασύνδεση, τα διαχειριζόμαστε μέσω του JBuilder, ενώ τα αρχεία της κάρτας τα επεξεργαζόμαστε μέσα από το JBuilder αλλά τα μεταγλωττίζουμε μέσω της γραμμής εντολών με τον μεταγλωττιστή της java. Παρόλα αυτά ο φάκελος αυτός είναι ένα project του JBuilder. Ο πηγαίος κώδικας βρίσκεται στο φάκελο `C:\JCard_APDU_sim\src` όπου εκεί διακρίνουμε τέσσερα πακέτα, το `purseclient`, το `Bonusclient`, το

purse και το Bonus. Στο φάκελο C:\JCard_APDU_sim\classes βρίσκονται τα μεταγλωττισμένα αρχεία της εφαρμογής. Χάριν ευκολίας μπορούμε να προσθέσουμε στο environment.bat μία ακόμα γραμμή, την set APLT=C:\JCard_APDU_sim η οποία θα μας διευκολύνει στην αυτοματοποίηση ορισμένων διαδικασιών, όπως η μεταγλώττιση και η μετατροπή σε κώδικα κάρτας.

Η διαδικασία παραγωγής κώδικα κάρτας

Αφού έχουμε γράψει τον πηγαίο κώδικα, είμαστε έτοιμοι να τον μεταγλωττίσουμε. Πριν το κάνουμε αυτό πρέπει να θέσουμε κάποιες μεταβλητές περιβάλλοντος ώστε να κατευθύνουμε τον μεταγλωττιστή της java ώστε να είναι σύμφωνος με τις δικές μας ανάγκες. Έτσι θέτουμε τις μεταβλητές:

```
set JCFLAGS=-g -d %APLT%\classes -classpath %JC_PATH%
set APLT=C:\JCard_APDU_sim (εάν δεν την έχουμε θέσει ήδη)
set JC_PATH=.;%APLT%\classes;%JC_HOME%\lib\api.jar
```

Τώρα είμαστε έτοιμοι να μεταγλωττίσουμε τον κώδικά μας με την εντολή

```
javac %JCFLAGS% src\purse\*.java
```

η οποία θα εκτελεστεί από το directory C:\Purse. Η παραπάνω εντολή μεταγλωττίζει όλα τα αρχεία πηγαίου κώδικα μόνο του πακέτου purse. Για να μεταγλωττίσουμε και τα υπόλοιπα πακέτα μπορούμε να προσθέσουμε στην εντολή τα αρχεία του πακέτου που θέλουμε να μεταγλωττίσουμε, ή επαναλαμβάνοντας την αντίστοιχη διαδικασία.

Σε αυτό το σημείο είμαστε έτοιμοι να παράγουμε αρχεία κώδικα κάρτας, να κάνουμε το λεγόμενο conversion. Αυτή η διαδικασία θα πραγματοποιηθεί με το εργαλείο converter, και πιο συγκεκριμένα με κλήση του αρχείου converter.bat που βρίσκεται στο %JC_HOME%\bin. Στην ουσία ο converter βρίσκεται στην διαδρομή %JC_HOME%\lib, που βρίσκονται όλες οι βιβλιοθήκες, αλλά καλείται αυτοματοποιημένα από το converter.bat. Η σύνταξη του ακολουθεί τον εξής τύπο:

```
converter –(options)
```

όπου στις επιλογές για το δικό μας project μας ενδιαφέρουν οι:

```
-out EXP JCA CAP
```

δήλωση των αρχείων που θέλουμε να παράγει η διαδικασία του convertion

```
-exportpath (PATH)
```

η διαδρομή στην οποία υπάρχουν τα αναγκαία export files

```
-applet appletAID packagename.classname
packagename packageAID version
```

ορισμός των AID του applet και του πακέτου που το περιέχει αλλά και ορισμός της έκδοσης αυτών

Για την γρηγορότερη διεκπεραίωση της διαδικασίας μπορούμε να ομαδοποιήσουμε όλες αυτές τις επιλογές σε ένα αρχείο με επέκταση .opt, το οποίο θα διοχετεύεται στον converter με το όρισμα –config και έπειτα το όνομα του αρχείου. Κάθε ένα τέτοιο αρχείο είναι υπεύθυνο για την παραγωγή μόνο ενός αρχείου με κώδικα κάρτας (.cap) Το αρχείο αυτό όμως αν και μπορεί να βρίσκεται οπουδήποτε στον δίσκο, η διαδρομή από όπου θα τρέξουμε τον converter είναι πάντοτε συγκεκριμένη. Δεδομένου του ότι τα μεταγλωττισμένα αρχεία βρίσκονται στον φάκελο `%APLT%\classes` και τα περιεχόμενα του opt αρχείου μας είναι τα:

```
-out EXP JCA CAP
-exportpath .
-applet            0xa0:0x0:0x0:0x0:0x62:0x3:0x1:0xc:0x8:0x1
purse.PurseApplet
purse 0xa0:0x0:0x0:0x0:0x62:0x3:0x1:0xc:0x8 1.0
```

Ο converter σε αυτήν την περίπτωση πρέπει να εκτελεστεί στην διαδρομή %APLT %\classes ώστε να μπορέσει να βρει το πακέτο purse που ορίζεται από το αρχείο opt. Συνήθως τα opt αρχεία τοποθετούνται στον ίδιο φάκελο με τον πηγαίο κώδικα. Έτσι στην δικιά μας εφαρμογή εκτελούμε από την γραμμή εντολών, στην διαδρομή %APLT%\ classes, την εντολή:

```
converter –config ..\src\purse\purse.opt
```

Στο σημείο αυτό πρέπει να κάνουμε μερικές σημαντικές παρατηρήσεις για την προηγούμενη διαδικασία. Αν έχουμε να κάνουμε με μία RMI εφαρμογή θα πρέπει κάποια στιγμή να δημιουργήσουμε ένα αρχείο class τύπου Stub, το οποίο είναι απαραίτητο για να καταστεί εφικτή η επικοινωνία μεταξύ των εφαρμογών. Αυτό πρέπει να γίνει οπωσδήποτε μετά από την παραγωγή του κώδικα κάρτας, καθώς αν γίνει πριν ο converter θα λάβει υπ' όψη του και αυτό το αρχείο στην μετατροπή, πράγμα που δεν είναι επιθυμητό και οδηγεί σε σφάλματα στην διαδικασία. Αν πρόκειται για APDU εφαρμογή δε χρειάζεται να δημιουργήσουμε κάποιο επιπλέον αρχείο. Δεύτερη παρατήρηση που πρέπει να κάνουμε, είναι ότι στο opt αρχείο δεν μπορούμε να ορίσουμε διαδρομές με μεταβλητές περιβάλλοντος. Έτσι θα πρέπει κάθε φορά να γράφουμε ολόκληρο το μονοπάτι. Τέλος όπως έχουμε ξαναπεί, πολύ σημαντική θέση σε αυτήν την διαδικασία έχουν τα export files. Όταν αναζητούμε ένα τέτοιο αρχείο, δεν πρέπει να το διοχετεύουμε στον converter δείχνοντας απ' ευθείας σε αυτό, αλλά δείχνοντας στο πακέτο που το περιέχει. Στην παρούσα εφαρμογή χρησιμοποιούμε export files της έκδοσης Java Card 2.1.1.. Επιπλέον λόγο του ότι έχει σχεδιαστεί να συνυπάρχουν δύο εφαρμογές μέσα στην κάρτα, γίνεται πρώτα μετατροπή της εφαρμογής που υλοποιεί τη διαμοιραζόμενη διασύνδεση, όπως η purse εφαρμογή, από την οποία μετατροπή παράγεται και ένα export αρχείο το οποίο αναφέρεται σε αυτήν. Αυτό το αρχείο χρησιμοποιείται από κάθε εφαρμογή η οποία προτίθεται να χρησιμοποιήσει τη διαμοιραζόμενη διασύνδεση αυτή, στη συγκεκριμένη περίπτωση η bonus εφαρμογή, πάντα τη στιγμή της μετατροπής.

Αφού κάναμε όλα τα απαραίτητα βήματα, είμαστε πλέον στην θέση να φορτώσουμε την εφαρμογή στην κάρτα. Υπάρχει όμως και η δυνατότητα να

ακολουθήσουμε τον δρόμο της προσομοίωσης. Για να το κάνουμε αυτό όμως πρέπει να εκτελέσουμε λίγα ακόμα βήματα.

Η εκτέλεση των εκτός κάρτας εφαρμογών

Στην πλειονότητα των περιπτώσεων, όπως αναφέραμε και παραπάνω, μαζί με την εφαρμογή της κάρτας αναπτύσσονται μαζί και άλλες εφαρμογές οι οποίες θα αλληλεπιδρούν με αυτήν. Στο συγκεκριμένο έργο αναπτύσσονται δύο εφαρμογές οι οποίες επικοινωνούν με την κάρτα, μία για κάθε εφαρμογή της κάρτας.

Η εφαρμογή πελάτη μπορεί να γραφτεί σε κάποιο IDE, όπως και κάθε κλασσική εφαρμογή, και έτσι μπορούμε πολύ εύκολα να προσθέσουμε γραφικό περιβάλλον και να έχουμε όλες τις ανέσεις που μας προσφέρει αυτό. Σημαντικό στοιχείο για να επικοινωνήσουν οι δύο εφαρμογές είναι η OpenCard τεχνολογία, και ένα στοιχείο για το οποίο πρέπει να μεριμνήσουμε είναι το αρχείο με την ονομασία opencard.properties. Αυτό παρέχει τις οδηγίες στην εφαρμογή πελάτη για το πώς θα συνδεθεί με την συσκευή που διαχειρίζεται την κάρτα, είτε αυτήν πρόκειται για την «συσκευή» προσομοίωσης, (cref), είτε πρόκειται για μία πραγματική συσκευή. Αυτό το αρχείο παρέχεται από την sun για το cref ή από τον προμηθευτή των καρτών. Η εφαρμογή πελάτη αναζητά το αρχείο όταν θέλει να συνδεθεί με την κάρτα, το οποίο σε αυτό το project πρέπει να βρίσκεται στην διαδρομή `C:\`.

Τέλος για να εκτελεστούν επιτυχώς, πρέπει να είναι παρόν στον ίδιο κατάλογο οι βιβλιοθήκες που χρησιμοποιεί, οι οποίες είναι οι:

```
apduio.jar
base-core.jar
base-opt.jar
javacardframework.jar
jcclientsamples.jar
jcrmiclientframework.jar
```

Οι εφαρμογές έχουν συμπιεστεί σε δυο jar αρχεία και είναι εκτελέσιμες σε όλα τα υπολογιστικά συστήματα, αρκεί να υπάρχει το κατάλληλο JRE.

Η μετατροπή σε jar γίνεται με τις εντολές, από τον φάκελο classes:

```
jar -cmf MANIFESTP.MF PurseClient.jar purseclient\*.class purseclient\6-pin.gif

jar -cmf MANIFESTB.MF BonusClient.jar BonusClient\*.class BonusClient\6-pin.gif
```

Προσομοίωση

Για να εργαστούμε σε περιβάλλον προσομοίωσης μπορούμε να χρησιμοποιήσουμε τα δυο εργαλεία που μας προσφέρει το JCDK, τα οποία είναι το jcwde και το cref, όπως είπαμε παραπάνω. Στην δική μας περίπτωση προτιμάμε να προσομοιώσουμε το περιβάλλον της κάρτας με το cref, καθώς προσεγγίζει την πραγματικότητα καλύτερα από το jcwde. Το cref μπορεί να εκτελεστεί με διάφορες εντολές, από τις οποίες οι δύο βασικότερες είναι αυτές για εγγραφή και ανάγνωση, καθώς οι υπόλοιπες έχουν σαν στόχο την βοήθεια των προγραμματιστών για την αποσφαλμάτωση της εφαρμογής. Με την εντολή

```
cref –o purse
```

ανοίγεται το αρχείο με όνομα purse για εγγραφή, ώστε να αποθηκευτεί σε αυτό μια «εικονική» EEPROM στην οποία θα έχουμε αποθηκεύσει κάποια δεδομένα. Αν το αρχείο δεν υπάρχει δημιουργείται, ενώ εάν προϋπάρχει διαγράφονται τα δεδομένα του χωρίς προειδοποίηση. Αφού έχουμε δημιουργήσει το αρχείο μπορούμε να αναγνώσουμε τα δεδομένα που έχουμε αποθηκεύσει σε μία τέτοια «εικονική» μνήμη με την εντολή:

```
cref -i purse
```

Οποιεσδήποτε μεταβολές γίνονται κατά την διάρκεια της ανάγνωσης δεν αποθηκεύονται. Σε κάθε περίπτωση όταν εκδίδουμε αυτή την εντολή το αρχείο πρέπει να υπάρχει και να βρίσκεται στην διαδρομή που εκτελείται η εντολή. Τέλος μπορούμε να

έχουμε και έναν συνδυασμό των παραπάνω μεθόδων, δηλαδή μπορούμε να ανοίξουμε μία εικονική μνήμη για ανάγνωση, να κάνουμε κάποιες τροποποιήσεις πάνω σε αυτή, και να αποθηκεύσουμε το αποτέλεσμα, δηλαδή την νέα εικονική μνήμη. Αυτό μπορούμε να το πετύχουμε με την εντολή:

```
cref -o purse2 –i purse
```

Έτσι μπορούμε να αποθηκεύσουμε τις μεταβολές που έγιναν στο αρχείο purse, στο purse2. Μπορούμε όμως να χρησιμοποιήσουμε και το ίδιο αρχείο και για ανάγνωση αλλά και για εγγραφή, δίνοντας την εντολή:

```
cref -o purse –i purse
```

Για να αποθηκεύσουμε τώρα δεδομένα σε μία εικονική μνήμη, ανοίγουμε ένα αρχείο για εγγραφή και έπειτα το cref περιμένει APDU εντολές σε μία θύρα TCP/IP η οποία εξ ορισμού είναι η 9025. Για να στείλουμε APDU εντολές χρησιμοποιούμε το εργαλείο apdutool, από ένα νέο παράθυρο, με την σύνταξη:

```
apdutool <APDU commands>
```

Βέβαια μπορούμε να ομαδοποιήσουμε τα APDU command σε ένα script αρχείο, το οποίο έπειτα θα διοχετεύσουμε στο apdutool. Αυτό γίνεται με την εντολή:

```
apdutool <APDU script file>
```

Ένα script αρχείο μπορεί να δημιουργηθεί είτε χειροκίνητα, είτε από το scriptgen. Όταν επιθυμούμε να φορτώσουμε ένα αρχείο με κώδικα κάρτας στην εικονική μνήμη (cap file), πρέπει να το μετατρέψουμε σε ένα script αρχείο, δηλαδή σε APDU’s, και έπειτα να κάνουμε κάποιες απαραίτητες προσθήκες στην αρχή και στο τέλος αυτού. Πρέπει να προστεθούν τα εξής:

- ❖ Στην αρχή τα περιεχόμενα του αρχείου Header.scr που περιέχει την εντολή powerup, ώστε να ξεκινήσει η λειτουργία της κάρτας
- ❖ Στο τέλος μία εντολή που θα δημιουργεί το στιγμιότυπο στην κάρτα και έπειτα τα περιεχόμενα ενός αρχείου Footer.scr που περιέχει την εντολή powerdown, για διακοπή της λειτουργίας της κάρτας.
- ❖ Τα αρχεία Header.scr και Footer.scr παρέχονται από το JCDK, ενώ η εντολή δημιουργίας του στιγμιότυπου θα εισαχθεί χειροκίνητα. Για αυτοματοποίηση της διαδικασίας μπορούμε να αποθηκεύσουμε την τελευταία εντολή σε ένα αρχείο με όνομα Create.scr. Για την απλοποίηση των διεργασιών τα περιεχόμενα του Create αρχείου έχουν εισαχθεί στο Footer.

Συγκεκριμένα εκτελούμε:

```
scriptgen -o %APLT%\scr\purse.scr %APLT
%\classes\purse\javacard\purse.cap

copy /b %APLT%\Header.scr+%APLT%\purse.scr +%APLT
%\Create.scr+%APLT%\Footer.scr %APLT%\purse_final.scr

apdutool %APLT%\purse_final.scr
```

Στο σημείο αυτό έχουμε δημιουργήσει ένα script αρχείο που περιέχει τον κώδικα κάρτας σε μορφή APDU's και κάποιες επιπλέον πληροφορίες, και έπειτα στο στείλουμε στην εικονική μνήμη και εγκαθίσταται σε αυτήν. Εκείνη την στιγμή με λίγα λόγια εκτελείται η μέθοδος install του applet μας. Πλέον έχουμε μία εικονική EEPROM που περιέχει ένα πακέτο λογισμικού, το οποίο εμείς έχουμε τοποθετήσει. Από αυτή τη στιγμή μπορούμε να διαβάσουμε την εικονική αυτή μνήμη, να μεταβάλλουμε τα στοιχεία της (στο ηλεκτρονικό πορτοφόλι για παράδειγμα να προσθέσουμε ή να αφαιρέσουμε χρηματικές μονάδες), ή να τα αφήσουμε αμετάβλητα, τις περισσότερες φορές με την μεσολάβηση μίας εφαρμογής πελάτη. Στην περίπτωση που προείπαμε, ότι μπορούμε να

κάνουμε κάποιες μεταβολές, περιλαμβάνεται και η περίπτωση να προσθέσουμε (ή να αφαιρέσουμε) κάποιο applet, το οποίο μπορεί και να αλληλεπιδρά με το υπάρχον.

Όλες αυτές οι εντολές περιέχονται στο αρχείο ServerPurse.bat για την εφαρμογή του ηλεκτρονικού πορτοφολιού, αλλά αυτό πρέπει να εκτελεστεί αφού πριν έχει τρέξει σε ένα άλλο παράθυρο η cref –i <όνομα εικόνας>. Παρόμοια οι παραπάνω διαδικασίες γίνονται και για την Bonus εφαρμογή, οι οποίες βρίσκονται στο ServerBonus.bat.

Μέρος Δεύτερο:
Στατική Ανάλυση
Προγραμμάτων Java Card

Στατική ανάλυση

Εισαγωγή

Οι έξυπνες κάρτες είναι μια τεχνολογία η οποία, όπως είπαμε, χρησιμοποιείται πολλές φορές για κρίσιμες εφαρμογές που απαιτούν υψηλό επίπεδο ασφάλειας, όπως για παράδειγμα σε τραπεζικές συναλλαγές. Λόγω αυτής της ιδιαιτερότητας, θα ήταν πολύ χρήσιμο, για αυτόν που αναπτύσσει εφαρμογές σε έξυπνες κάρτες, ένα εργαλείο το οποίο θα είναι ικανό να ελέγχει τον κώδικα και να ανακαλύπτει αδυναμίες αυτού. Γενικά, για να επιτευχθεί ένας τέτοιος σκοπός, ο έλεγχος του προγράμματος από άλλο πρόγραμμα, πρέπει να χρησιμοποιηθούν εργαλεία που είτε ελέγχουν τον κώδικα καθ αυτόν είτε δρούν σαν ένα ενδιάμεσο στρώμα (*layer*) μεταξύ μηχανής και προγράμματος ώστε να γίνει έλεγχος σε κανονικό χρόνο εκτέλεσης. Στην τελευταία κατηγορία γίνεται δυναμική ανάλυση του προγράμματος, ενώ στην πρώτη η οποία θα αναλυθεί παρακάτω, στατική ανάλυση. Η στατική ανάλυση με την σειρά της χωρίζεται σε δύο κατηγορίες, την στατική ανάλυση προγράμματος (*static program analysis*) και τον έλεγχο μοντέλων (*model checking*).

Στην συγκεκριμένη εργασία θα ασχοληθούμε με την στατική ανάλυση προγράμματος, η οποία αν και υστερεί σε αρκετά σημεία έναντι των άλλων αναλύσεων που αναφέραμε, όπως στην ακρίβεια της ανάλυσης, παρόλα αυτά υπερτερεί σε άλλους τομείς, όπως στην ταχύτητα ανάλυσης.

Το εργαλείο που χρησιμοποιήθηκε ώστε να εφαρμοστεί η στατική ανάλυση προγράμματος σε προγράμματα έξυπνων καρτών είναι το FindBugs, ένα εργαλείο ανοιχτού κώδικα απο το πανεπιστήμιο του Maryland το οποίο αναλύει στατικά προγράμματα γραμμένα σε Java. Λόγω της φύσης του εν λόγω προγράμματος, του γεγονότος δηλαδή ότι είναι ανοιχτού κώδικα, μας δόθηκε η δυνατότητα να αναγνώσουμε τον κώδικα του και να γράψουμε άλλο κώδικα, ο οποίος θα εκτελείται ώς πρόσθετος (*plugin*) σε αυτό. Παρακάτω θα περιγραφεί η διαδικασία συγγραφής του κώδικα, καθώς και λεπτομέρειες για το έργο που επιτελεί.

Στατική Ανάλυση Προγράμματος

Πριν ασχοληθούμε με την λειτουργία του FindBugs, είναι σημαντική η αναφορά στην στατική ανάλυση, ώστε να υπάρχει μια πληρότητα σχετικά με το θέμα που θα αναλυθεί. Με την στατική ανάλυση, έχουμε την δυνατότητα να επιθεωρήσουμε τον κώδικα που παράγουμε στατικά, χωρίς δηλαδή να τον εκτελέσουμε. Βέβαια, υπάρχουν διαφόρων τύπων εργαλεία που εφαρμόζουν στατική ανάλυση σε διαφορετικά επίπεδα, όπως για παράδειγμα στην Java σε επίπεδο καθαρού κώδικα και σε επίπεδο ενδιάμεσου κώδικα (bytecode). Το κοινό όμως στοιχείο των διαφορετικών προϊόντων λογισμικού είναι ότι δεν εκτελούν το πρόγραμμα αλλά αναζητούν σε αυτόν κάποια πρότυπα (patterns) τα οποία είναι ύποπτα για την εμφάνιση σφαλμάτων (*bug*).

Είναι εύκολο να εξαχθεί το συμπέρασμα, ότι ο σημαντικότερος κώδικας μέσα σε ένα τέτοιο πρόγραμμα είναι τα πρότυπα με τα οποία έχει εφοδιαστεί ώστε να γίνεται η ανίχνευση των σφαλμάτων. Τα πρότυπα αυτά, πρέπει να περιγράφουν με όση μεγαλύτερη ακρίβεια το πώς "μοιάζει" ένα κομμάτι κώδικα το οποίο είναι επίφοβο για την εμφάνιση σφάλματος, ώστε να αποφευχθεί μια τέτοια περίπτωση. Αρκετές όμως φορές υπάρχει η πιθανότητα, λόγω φτωχών προτύπων, να εμφανιστεί κάποιο κομμάτι κώδικα ως επικίνδυνο ενώ στην πραγματικότητα δεν είναι. Συμβαίνει όμως και το αντίθετο για τον ίδιο λόγο, η αποτυχία εύρεσης του επίφοβου κώδικα. Στην πρώτη περίπτωση λέγεται ότι η εκτέλεση του αναλυτή μπορεί να μας επιστρέψει μη έγκυρα θετικά αποτελέσματα (*false positives*), ενώ στην δεύτερη δεν εμφανίζονται τα έγκυρα αρνητικά αποτελέσματα (*false negatives*).

Συνήθως, για να φτάσει ένα πρότυπο να επιστρέφει μόνο έγκυρα αποτελέσματα, χρειάζεται μεγάλο κόπο, και έτσι θεωρείται καλύτερο τα αποτελέσματα που λαμβάνουμε να περιέχουν μερικά μη έγκυρα θετικά αποτελέσματα, τα οποία μπορούν εύκολα να διαπιστωθούν ότι δεν είναι έγκυρα, παρά να μην εμφανίζονται τα πραγματικά κομμάτια κώδικα που έχουν πρόβλημα, εφόσον δεν είναι δυνατόν να επιστρέφονται μόνο τα πραγματικά προβλήματα. Επίσης, λαμβάνοντας υπόψιν ότι πολυπλοκότητα στα πρότυπα σημαίνει και επιβράδυνση της ανάλυσης που επιτελείται, οδηγούμαστε στο συμπέρασμα ότι πρέπει να υπάρξει κάποιος συμβιβασμός μεταξύ της ποιότητας των προτύπων και της ταχύτητας της ανάλυσης. Συνοψίζοντας, οι παράγοντες που χαρακτηρίζουν ένα πρότυπο είναι η πολυπλοκότητα του, ο βαθμός ολοκλήρωσης και η ταχύτητα της ανάλυσης.

Πιο ειδικά, τα πρότυπα οδηγούν τον αναλυτή δίνοντας του ειδικές πληροφορίες για το πώς είναι ένα σφάλμα, όπως για παράδειγμα κάποια συγκεκριμένη αλληλουχία κώδικα που είναι επίφοβη και οι οντότητες που τα υλοποιούν ονομάζονται ανιχνευτές (*detectors*). Είναι επίσης πιθανό να δίνει την πληροφορία ποιες τιμές κάποιων μεταβλητών είναι δυνατόν να δημιουργήσουν πρόβλημα και να ερευνάται στην συνέχεια αν κάτι τέτοιο ισχύει στατικά. Μία ακόμη περίπτωση είναι να δίνει στοιχεία που έχουν σχέση με την ροή εκτέλεσης του προγράμματος και να γίνεται ανάλυση σε αυτήν την κατεύθυνση. Έτσι, υπάρχουν αρκετών ειδών ανιχνευτές, τα οποία χωρίζονται σε μεγάλες κατηγορίες όπως αυτά που εφαρμόζονται για

- συγκεκριμένες αλληλουχίες κώδικα (*Code or bytecode scanning detectors*)
- έλεγχο ροής έλέγχου (*Control flow detectors*)
- έλεγχο ροής δεδομένων (*Dataflow detectors*)

Στο σημείο αυτό να σημειωθεί ότι ένας αναλυτής μπορεί να λάβει υπόψη του σε μια εκτέλεση πολλά από αυτά τα πρότυπα και να εφαρμόσει δηλαδή όλους τους αντίστοιχους ελέγχους.

Για να γίνει πιο κατανοητός ο ρόλος του προτύπου, θα εφαρμόσουμε στο κομμάτι κώδικα που απεικονίζεται στην εικόνα ένα απλό πρότυπο. Οι δυο γραμμές αυτές κώδικα είναι απόσπασμα από την beta έκδοση του `Java Development Kit 1.6.0`, και περιέχουν ένα απλό σφάλμα: γίνεται έλεγχος αν μια μεταβλητή είναι κενή (*null*) και σε περίπτωση που είναι, χρησιμοποιείται. Θα μπορούσαμε σε αυτό το σημείο να αναλύσουμε στατικά τον κώδικα δίνοντας του την εξής πληροφορία: “Αναζήτησε στον δεδομένο κώδικα αν κάποια μεταβλητή που είναι κενή χρησιμοποιείται”. Αν τώρα θα θέλαμε να κάνουμε το πρότυπο μας πιο εξειδικευμένο θα μπορούσαμε να πούμε: “Αναζήτησε στον δεδομένο κώδικα αν κάποια μεταβλητή που ελέγχεται αν είναι κενή, χρησιμοποιείται μέσα στο μπλοκ του κώδικα που γίνεται ο έλεγχος”. Για τον συγκεκριμένο κώδικα και τα δυο πρότυπα είναι έγκυρα, αλλά το δεύτερο είναι πιο ειδικό και πρέπει να δαπανηθεί περισσότερος κόπος για την κωδικοποίηση του.

```
if (listeners == null)
    listeners.remove(listener);
```

Ενδεικτικά κάποιοι έλεγχοι που μπορούν να επιτελέσουν οι ανιχνευτές είναι οι ακόλουθοι:

- Έλεγχος για πιθανές υπερχειλίσεις (*Overflow*)
- Αναζήτηση για παράτυπη χρήση μεθόδων (*Illegal use*)
- Εύρεση πιθανού κακόβουλου κώδικα (*Malicious code*)
- Εύρεση κώδικα που υιοθετεί κακές πρακτικές προγραμματισμού (*Bad practice*)
- Ανακάλυψη προβλημάτων σε πολυνηματικά προγράμματα (*thread bugs*)
- Έλεγχος ορίων τιμών μεταβλητών (*illegal values*)

Το FindBugs

Το FindBugs είναι ένα εργαλείο ανοιχτού κώδικα που προέρχεται από το πανεπιστήμιο του Maryland, το οποίο επιτρέπει να αναλύουμε στατικά προγράμματα γραμμένα σε Java. Δημιουργοί του είναι ο Bill Pugh και ο David Hovemeyer απο το πανεπιστήμιο του Maryland και έχουν θέσει σε διαθεσιμότητα το εν λόγω πρόγραμμα δωρεάν, υπό την άδεια GPL (GNU Public Licence). Σαν έργο ανοιχτού κώδικα, φιλοξενείται από τον διαδικτυακό τόπο SourceForge, ο οποίος είναι ένας από τους πλέον πιο δημοφιλής τόπους για την φιλοξενία έργων ανοιχτού λογισμικού, και είναι ένα ενεργό έργο το οποίο προσφέρει αρκετά συχνά νέες βελτιωμένες εκδόσεις. Ο κώδικας του είναι γραμμένος σε Java, γεγονός που το κάνει εκτελέσιμο σε διαφορετικές πλατφόρμες και ενθαρρύνει τα άτομα τα οποία επιθυμούν να συγγράψουν κώδικα για αυτό, λόγω της μεγάλης διάδοσης του αντικειμενοστραφούς προγραμματισμού και ποιο συγκεκριμένα της Java.

Γενικός σκοπός του FindBugs είναι η γρήγορη (στατική) ανάλυση προϊόντων λογισμικού, που όπως είπαμε είναι κωδικοποιημένα σε Java, εφαρμόζοντας ένα μεγάλο ρεπερτόριο από πρότυπα, υλοποιημένα με τους κατάλληλους ανιχνευτές. Οι ανιχνευτές αυτοί έχουν υλοποιηθεί με τρόπο ώστε να είναι όσο πιο κοντά γίνεται στην χρυσή τομή μεταξύ της ταχύτητας και της ακρίβειας, δίνοντας μεγάλη έμφαση στην ταχύτητα, ώστε να λαμβάνει ο χρήστης σε μικρό χρονικό διάστημα όσο πιο έγκυρα αποτελέσματα είναι δυνατόν. Έτσι, σε αντίθεση με άλλους αναλυτές που είναι χρονοβόροι και παρέχουν μεγάλη ακρίβεια, το FindBugs κάνει μια σύμβαση στην ποιότητα επενδύοντας παράλληλα στην οικονομία χρόνου. Έτσι λόγω της ταχύτητας και της ανοιχτής του

φύσης το FindBugs έχει κερδίσει αρκετούς υποστηρικτές, από την επιστημονική κοινότητα και όχι μόνο.

Σημαντικό είναι να τονίσουμε ότι παρόλα τα θετικά που προσφέρει, σαν προϊόν ανοιχτού κώδικα, το FindBugs υστερεί στην τεκμηρίωση, τόσο σε επίπεδο κώδικα και σχολίων, όσο και περιγραφής της λειτουργίας του. Υπάρχει δηλαδή ανεπάρκεια στα σχόλια μέσα στον κώδικα και η λειτουργία του εξηγείται από ένα έγγραφο το οποίο συνοδεύει τον πηγαίο κώδικα. Το έγγραφο αυτό (*architecture.tex*) είναι προσανατολισμένο στην κατασκευή νέων ανιχνευτών, εξηγώντας επιφανειακά την αρχιτεκτονική του και τα είδη των ανιχνευτών που είναι δυνατόν να υλοποιηθούν.

Λειτουργία

Περιβάλλον χρήσης

Ξεκινώντας από την διασύνδεση με τον χρήστη, το FindBugs παρέχει την δυνατότητα εκτέλεσης με γραφικό περιβάλλον (*GUI – Graphical User Interface*), το οποίο είναι κατάλληλο για αρχάριους χρήστες, αλλά και την δυνατότητα εκτέλεσης μέσω κονσόλας (*TUI – Text User Interface*). Όπως είναι αναμενόμενο, η εκτέλεση του FindBugs μέσω γραμμής εντολών προσφέρει ταχύτητα στην ανάλυση, αλλά χρειάζεται εξοικείωση από μέρος των χρηστών. Αντιθέτως, το γραφικό περιβάλλον αν και προσφέρει μια πιο αργή εκτέλεση, είναι φιλικό και εμφανίζει τα αποτελέσματα ομαδοποιημένα με λογικό τρόπο, καθώς επίσης προσφέρει και άμεση οπτική αναπαράσταση του κώδικα στον οποίο εμφανίστηκε το σφάλμα. Το γραφικό περιβάλλον είναι αρκετά λιτό και απέριττο και υπάρχει η δυνατότητα της ομαδοποίησης των σφαλμάτων όπως επιθυμεί ο χρήστης.

Δυνατότητες

Ανεξάρτητα με το γραφικό περιβάλλον όμως, ο χρήστης δεν έχει πολλές επιλογές σχετικά με την διαχείριση της λειτουργίας του FindBugs ανεξάρτητα από την διασύνδεση που επιλέγει. Οι κύριες διεργασίες που μπορεί να επιτελέσει είναι οι εξής:

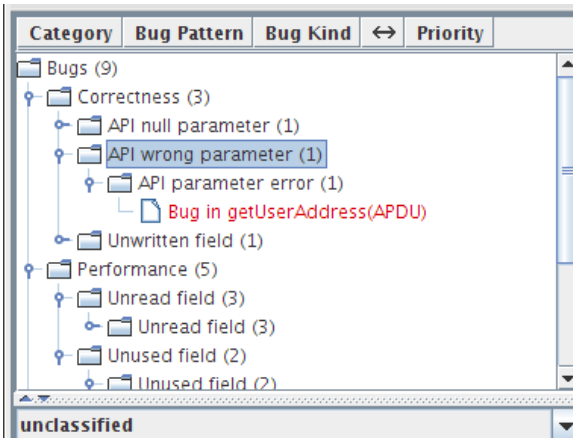
- Δημιουργία νέου έργου ανάλυσης
- Αποθήκευση των αποτελεσμάτων μετά από την ανάλυση
- Άνοιγμα ενός αποθηκευμένου έργου

- Αλλαγή σειράς εμφάνισης των κατηγοριών σφαλμάτων
- Ορισμός φίλτρων σφαλμάτων
- Ανάθεση σχολίων και ειδικών κατηγοριών στα σφάλματα
- Πλοήγηση στο κώδικα του προγράμματος που αναλύεται

Τα τρία πρώτα είναι κλασσικά χαρακτηριστικά που έχουν τα περισσότερα προγράμματα, ώστε να διευκολύνουν τον χρήστη στην αποθήκευση και την ανάκληση των συνεδριών που κατασκευάζονται από τον χρήστη. Μια συνεδρία περιέχει στοιχεία του έργου, όπως τις διαδρομές του πηγαίου κώδικα και των μεταγλωττισμένων αρχείων, καθώς και καταγραφές των σφαλμάτων που βρέθηκαν κατά την διάρκεια της ανάλυσης. Όλα αυτά, αποθηκεύονται σε ένα αρχείο, ακολουθώντας μια καθορισμένη μορφή που υιοθετεί το FindBugs με την γλώσσα *Extensible Markup Language* (*XML*). Έτσι είναι πολύ εύκολο στον χρήστη να αποθηκεύσει και να ανακαλέσει τις συνεδρίες που δημιούργησε, μαζί με τα σφάλματα που βρέθηκαν, είτε για ανάγνωση είτε για νέα ανάλυση, εφόσον έχει τροποποιηθεί ο κώδικας της εφαρμογής η οποία τίθεται προς ανάλυση.

Σχετικά με τις ενέργειες που μπορούν να εφαρμοστούν στα αποτελέσματα της ανάλυσης, όπως είπαμε και πιο πριν, ο χρήστης έχει περιορισμένες δυνατότητες. Υπάρχουν κάποιες κατηγορίες σφαλμάτων στις οποίες ανήκουν τα σφάλματα που ανακαλύπτει το FindBugs. Ένα σφάλμα μπορεί να λάβει μια προκαθορισμένη τιμή για κάθε μία κατηγορία. Οι κατηγορίες σφαλμάτων είναι εξής:

- *Category*
- *Bug Pattern*
- *Bug Kind*
- *Priority*



Η κατηγορία Category αντιπροσωπεύει την φύση του σφάλματος, όπως αυτό έχει επίπτωση στο έργο. Μπορεί να έχει διάφορες

τιμές, όπως *Correctness* (πρόβλημα ορθότητας) ή *Performance* (μη αποδοτικό κομμάτι κώδικα). Η *Bug Pattern* κατηγορία, δίνει πληροφορίες για το είδος του προτύπου το οποίο ανακάλυψε το σφάλμα αυτό και έχει τόσες τιμές όσες τα πρότυπα που υπάρχουν στο Findbugs, ενώ το *Bug Kind* συγκεκριμενοποιεί ακόμη περισσότερο το σφάλμα. Τέλος το Priority δίνει πληροφορίες για την προτεραιότητα του σφάλματος και μπορεί να είναι *High*, *Medium* ή *Low*. Υπάρχει μεγάλος αριθμός προκαθορισμένων τιμών σε κάθε κατηγορία, εκτός από την *Priority*, και εύκολα κανείς μπορεί να προσθέσει, μαζί με τα κατάλληλα πρότυπα, νέες κατάλληλες τιμές σε κάθε κατηγορία. Η απεικόνιση των σφαλμάτων γίνεται με δενδρική δομή, η οποία φαίνεται στο παραπάνω σχήμα, και ο χρήστης μπορεί να αλλάξει μόνο την σειρά εμφάνισης αυτών και να αναθέσει φίλτρα ώστε να μην εμφανίζονται σφάλματα συγκεκριμένων κατηγοριών.

Τέλος, ο χρήστης έχει την δυνατότητα να πλοηγηθεί στον κώδικα της εφαρμογής που αναλύεται και να γράψει σχόλια για κάθε σφάλμα που έχει βρεθεί. Επίσης μπορεί να κατηγοριοποιήσει τα σφάλματα, ώστε να γίνεται κατανοητό γρήγορα πως πρέπει να λυθεί.

Αρχιτεκτονική

Σε τεχνικό επίπεδο, το FindBugs χρησιμοποιεί διάφορες έτοιμες βιβλιοθήκες αλλά και δικό του κώδικα, ώστε να αναπαραστήσει και θα διαχειριστεί τα προγράμματα τα οποία θα αναλύσει. Μία βιβλιοθήκη η οποία είναι πολύ σημαντική για την εκτέλεση του είναι η *BCEL* (*ByteCode Engineering Library*), η οποία αναπαριστά τις δομές που υπάρχουν στην Java, όπως οι κλάσεις, οι μέθοδοι, οι μεταβλητές κλπ. Επομένως, λόγω του ότι η BCEL μας δίνει την δυνατότητα της αναπαράστασης αυτών των δομών, είναι δυνατή η ανίχνευση προτύπων μέσα σε αυτά μέσα από κατάλληλους ελέγχους. Εκτός όμως από την βιβλιοθήκη BCEL, το ίδιο το FindBugs υλοποιεί διάφορες δομές που θα χρειαστούν στην ανάλυση του κώδικα, όπως ο γράφος ελέγχου ροής δεδομένων, γνωστός ως *CFG* (*Control Flow Graph*).

Το FindBugs, για να αναλύσει τον κώδικα ενός προγράμματος, ελέγχει όλες τις κλάσεις και τις μεθόδους τους με όλα τα πρότυπα που υφίστανται, τα οποία υλοποιούνται από τους ανιχνευτές. Γενικά, οι ανιχνευτές που υπάρχουν μέσα στο FindBugs, υλοποιούν την τεχνική της επίσκεψης της κάθε μεθόδου, δηλαδή για να ελεγχθεί μια κλάση όλοι οι

ανιχνευτές πρέπει να ελέγξουν όλες τις μεθόδους της. Οι ανιχνευτές, έχουν πρόσβαση στα στοιχεία της εφαρμογής που αναλύεται, μέσω ορισμένων καθολικών δομών που ονομάζονται *αντικείμενα ανάλυσης* (*Analysis Objects*). Επίσης είναι δυνατόν να υπάρχουν περισσότερα από ένα *"περάσματα"* του κώδικα, καθώς μπορούνε να χρησιμοποιηθούν πληροφορίες σε ένα πέρασμα οι οποίες έχουν εξαχθεί από προηγούμενα περάσματα. Η τεχνική αυτή βασίζεται σε μια βάση δεδομένων που διατηρεί το FindBugs κατά την διάρκεια εκτέλεσης του, η οποία είναι προσπελάσιμη από τους ανιχνευτές. Επιπλέον, η βάση αυτή, μπορεί να χρησιμοποιηθεί για την διοχέτευση πληροφοριών από έναν ανιχνευτή σε έναν άλλο, ακόμα και στο ίδιο πέρασμα. Ειδικότερα, το *πλάνο εκτέλεσης* του FindBugs (*Execution Plan*), περιλαμβάνει πολλά *"περάσματα" ανάλυσης* (*AnalysisPasses*), όπου το καθένα περιέχει τους ανιχνευτές που θα χρησιμοποιηθούν και με ποια σειρά.

Συγκεκριμένα, τα βήματα που ακολουθεί το FindBugs είναι τα εξής:

1. Ανάγνωση του κώδικα του λογισμικού που τίθεται προς ανάλυση
2. Εύρεση όλων κλάσεων που ανήκουν στο έργο αυτό
3. Φόρτωση των διαθέσιμων plugin που περιέχουν τους ανιχνευτές
4. Δημιουργία του πλάνου εκτέλεσης
5. Εκτέλεση του αλγορίθμου του FindBugs, για την εφαρμογή των ανιχνευτών στις κατάλληλες κλάσεις

Στο τρίτο βήμα αναφέρεται ότι φορτώνονται τα διαθέσιμα plugins, το οποίο έχει σχέση με την αφαιρετική αρχιτεκτονική που υιοθετεί το FindBugs. Όλοι οι ανιχνευτές που υπάρχουν στο FindBugs, είναι πακεταρισμένοι σε ένα αρχείο *Java Archive* (*JAR*) με όνομα *coreplugin*, το οποίο είναι τοποθετημένο σε ένα κατάλογο με το όνομα `plugin`. Έτσι, αυτή η υλοποίηση βοηθάει όποιον θέλει να προσθέσει άλλους ανιχνευτές στο FindBugs, καθώς η μοναδική εργασία που πρέπει να γίνει, εκτός από την μεταγλώττιση του κώδικα και το πακετάρισμα σε ένα JAR αρχείο, είναι να αντιγραφεί το εν λόγω αρχείο – plugin στην διαδρομή που βρίσκεται το *coreplugin*. Το κάθε plugin, εκτός απο τον κώδικα που περιέχει, διαθέτει και τα μηνύματα και τις κατηγορίες των σφαλμάτων που μπορούν να ανακαλύψουν οι εν λόγω ανιχνευτές. Αυτά περιέχονται στα αρχεία

messages.xml και *findbugs.xml* (τα ονόματα είναι προκαθορισμένα), αντίστοιχα, τα οποία πρέπει να βρίσκονται στον κατάλογο με όνομα `etc`.

Οικογένειες ανιχνευτών

Μιλώντας πιο ειδικά, οι ανιχνευτές του FindBugs υπάγονται σε δύο κατηγορίες, ανάλογα με την ανάλυση που επιτελούν:

- *Visitor based*
- *Control Flow Graph based*

Στην πρώτη οικογένεια ανήκουν οι ανιχνευτές που πραγματοποιούν γραμμική σάρωση του bytecode, αναζητώντας για πρότυπα μέσα στον ενδιάμεσο κώδικα. Η λειτουργία των ανιχνευτών της συγκεκριμένης τάξης είναι στην ουσία η υλοποίηση ενός πεπερασμένου ντετερμινιστικού αυτόματου, το οποίο όταν φτάσει στην τελική κατάσταση σημαίνει ότι έχει βρεθεί ένα σφάλμα. Βέβαια, είναι πιθανό να υπάρχουν παραπάνω από μία τελικές καταστάσεις και συνεπώς ο ανιχνευτής να αναγνωρίζει πολλά διαφορετικά σφάλματα. Γενικά, αυτοί οι ανιχνευτές υλοποιούν την διασύνδεση (interface) `BytecodeScanningVisitor` και για να συγκεκριμενοποιηθεί η εργασία τους πρέπει να γίνει υπέρβαση (override) των αντίστοιχων μεθόδων που καλούνται, όταν για παράδειγμα γίνεται επίσκεψη σε μία μέθοδο ή όταν λαμβάνουμε την επόμενη γραμμή κώδικα, και να γραφεί ο κώδικας που αντιστοιχεί στο αυτόματο που έχει σχεδιαστεί. Χαρακτηριστικά της κατηγορίας είναι η ταχύτητα εκτέλεσης, καθώς δεν πρέπει να κατασκευαστούν πολύπλοκες δομές, αλλά και το ότι δεν είναι δυνατόν να ανακαλύψουμε σφάλματα με πολύπλοκη δομή, όπως σφάλματα που έχουν σχέση με νήματα. (π.χ. Races). Ένας σημαντικός ανιχνευτής που ανήκει σε αυτήν την οικογένεια, είναι ο `OpcodeStack`, ο οποίος προσπαθεί να κρατήσει πληροφορίες σχετικά με την *στοίβα τελεστών* (*operant stack*) και αποτελεί μια βάση για την εξέλιξη των ανιχνευτών, ώστε να γεφυρωθεί το χάσμα μεταξύ των δύο οικογενειών.

Στην δεύτερη οικογένεια, ανήκουν οι ανιχνευτές που χρησιμοποιούν τους γράφους ελέγχου ροής. Αυτοί δεν σαρώνουν τον κώδικα, ελέγχοντας κάθε γραμμή, αλλά εκμεταλλεύονται τις ιδιότητες των *μπλοκ κώδικα* (*basic blocks*), όπως την διάκριση των βρόγχων και σφάλματα που έχουν σχέση με την ροή του προγράμματος. Η

χαρακτηριστικότερη υποκατηγορία της CFG οικογένειας είναι οι ανιχνευτές που κάνουν έλεγχο ροής δεδομένων (*Data Flow Detectors*), και η λειτουργία τους βασίζεται πάνω στους *CFG*. Είναι πιο πολύπλοκοι απο τους *visitor based* ανιχνευτές, καθώς και πιο αργοί, αλλά προσφέρουν πιο πολλές δυνατότητες σχετικά με την ανεύρεση σφαλμάτων.

Στατική ανάλυση προγραμμάτων έξυπνων καρτών

Οι βιβλιοθήκες που προσφέρονται από την *Java Card* (*Java Card Application Program Interface – JC API*), παρέχουν ένα σύνολο από κλάσεις και διασυνδέσεις, οι οποίες αποκρύπτουν τις λεπτομέρειες της υλοποίησης τους, καθιστώντας λιγότερο επίπονο το έργο των προγραμματιστών. Έτσι, αυτό το γεγονός επιτρέπει την αφοσίωση τους στην ανάπτυξη των στοιχείων της εφαρμογής, υποθέτοντας την ορθή χρήση των μεθόδων του API. Η ορθή χρήση των μεθόδων του API μπορεί να σημαίνει την χρήση των κατάλληλων ορισμάτων (συμφωνία ως προς τον τύπο αλλά και ως προς τα όρια των τιμών) αλλά και την κλήση τους σε κατάλληλα σημεία του προγράμματος. Για παράδειγμα, υπάρχουν ορισμένες μέθοδοι που δεν μπορούν να δεχθούν αρνητικούς ακεραίους και άλλες οι οποίες πρέπει να κληθούν αυστηρά με μια συγκεκριμένη σειρά. Λόγω των χαρακτηριστικών των Java Card Applets και της δομής των πολλαπλών σημείων πρόσβασης των καρτών, είναι πιθανό μια εξαίρεση που μπορεί να συμβεί σε χρόνο εκτέλεσης να εμφανιστεί με την μορφή μηνύματος λάθους, αφενός να αποτελέσει πολύτιμη πληροφορία για κάποιον επιτιθέμενο και αφετέρου να αφήσει την εφαρμογή σε μία απρόβλεπτη και ευάλωτη, κατάσταση. Με βάση τα παραπάνω, αναπτύχθηκαν ορισμένοι ανιχνευτές, στο FindBugs, οι οποίοι έχουν την δυνατότητα να αναλύουν προγράμματα για έξυπνες κάρτες, γραμμένα σε Java Card, και να εποπτεύουν την ορθή χρήση των μεθόδων του API.

Εισαγωγή

Όπως προαναφέρθηκε μια εφαρμογή σε Java Card, ακολουθεί έναν κύκλο ζωής, οπού η εφαρμογή μπορεί να σε μία από τις φάσεις που υπάρχουν σε αυτόν. Αυτές οι φάσεις είναι οι εξής:

- *Φόρτωση*
- *Εγκατάσταση*
- *Προσωποποίηση*

- *Επιλέξιμη*
- *Μπλοκαρισμένη*
- *Νεκρή*

Χαρακτηριστικό των εφαρμογών Java Card είναι ότι σε κάθε φάση μπορούν να γίνουν συγκεκριμένες ενέργειες. Επίσης, αντίθετα με της εφαρμογές της Java, δεν υπάρχει ένα σημείο που θεωρείται σημείο εισόδου (μέθοδος main), αλλά πολλαπλά σημεία εισόδου τα οποία καλούνται από το JCRE όταν ληφθεί ένα πακέτο APDU.

Στην Java Card, κάθε εξαίρεση μπορεί να φτάσει μέχρι το υψηλότερο σημείο, το JCRE. Σε αυτήν την περίπτωση η εντολή που εκτελείται διακόπτεται και εκδίδεται ένα μήνυμα με την κατάλληλη λέξη σφάλματος (*Status Word*). Αν η εξαίρεση είναι τύπου *ISOException*, τότε μπορεί να αναγνωριστεί και να χρησιμοποιηθεί η κατάλληλη λέξη σφάλματος, αλλά σε άλλη περίπτωση εκδίδεται η λέξη λάθους με τιμή *0x6f0*, που σημαίνει ότι δεν υπάρχει ακριβής διάγνωση.

Μία εξαίρεση που θα φτάσει στο σημείο εισόδου της εφαρμογής, μπορεί να αποκαλύψει πληροφορίες σχετικά με την συμπεριφορά της, γεγονός που σε γενικές γραμμές απαγορεύεται. Στην πράξη, ενώ μια εξαίρεση τύπου *ISOException* εκδίδεται από τον κώδικα της ίδιας της εφαρμογή χρησιμοποιώντας την εντολή *throw*, μία πιθανή εξαίρεση που δεν χειρίστηκε σωστά προερχόμενη από την λανθασμένη χρήση μίας μεθόδου του API προκαλεί μη αναμενόμενο σφάλμα. Αυτό είναι ικανό να αφήσει την εφαρμογή σε μία απρόσμενη και ύπουλη κατάσταση η οποία μπορεί να παραβεί τα χαρακτηριστικά ασφάλειας της.

Λαμβάνοντας όλα τα προηγούμενα υπόψιν, χρησιμοποιούνται τα χαρακτηριστικά της ανάλυσης που μας προσφέρει το FindBugs, όπως η γραμμική σάρωση κώδικα και η χρήση των γράφων ελέγχου ροής, ελέγχονται ενδεικτικά η χρήση ορισμένων μεθόδων του API, οι οποίες μπορούν, μετά από εσφαλμένη χρήση, να οδηγήσουν στην μείωση της ασφάλειας της εφαρμογής.

Η συνεισφορά πίσω από αυτήν την εργασία είναι η απόδειξη του ότι είναι δυνατόν να ελέγξουμε με το συγκεκριμένο εργαλείο ανοικτού κώδικα την ορθή χρήση των μεθόδων του API της Java Card, γεγονός το οποίο μπορεί να οδηγήσει στον ευκολότερο έλεγχο των εφαρμογών από τους προγραμματιστές, εξοικονομώντας πολύτιμο χρόνο για την ανάπτυξη της εφαρμογής.

Ανιχνευτές

Προκειμένου να αναπτυχθούν οι ανιχνευτές, κρίθηκε αναγκαίο να αναπτυχθούν δύο εφαρμογές Java Card οι οποίες θα περιέχουν εσφαλμένη χρήση των μεθόδων του API και θα χρησιμοποιηθούν σαν βάση για πειραματισμό. Έτσι, υποθέτουμε ότι η μία εφαρμογή υλοποιεί ένα σύστημα ηλεκτρονικού πορτοφολιού το οποίο αλληλεπιδρά με μία άλλη εφαρμογή η οποία μπορεί να αποθηκεύει μονάδες επιβράβευσης (*bonus*), ανάλογα με την χρέωση της άλλης εφαρμογής. Η κάθε μια βρίσκεται σε διαφορετικό πακέτο και συνεπώς γίνεται υλοποίηση της διαμοιραζόμενης διασύνδεσης, όπως είδαμε στην σχετική ενότητα. Η κάθε εφαρμογή, έχει το δικό της ιδιωτικό κωδικό αναγνώρισης (*PIN*), και υπάρχει πρόσβαση σε κάθε μία από αυτές μέσω του *Java Card Runtime Environment* (*JCRE*), το οποίο καλεί την μέθοδο process για να εκτελεστούν οι διαθέσιμες λειτουργίες μετά από την κατάλληλη εντολή APDU.

Συγκεκριμένα οι δυο εφαρμογές έχουν τις εξής μεθόδους (πέρα από εκείνες που έχει κάθε Java Card εφαρμογή):

PurseApplet

- credit
- debit
- foreignDebit
- getAccountNumber
- getBalance
- getUsserAddress
- getUserName
- getUserSurname
- setAccountNumber
- setUserPIN
- setUsserAddress
- setUserName
- setUserSurname
- validateUserPIN

BonusApplet

- changeUserPIN
- eraseBonus
- getBonus
- makePurchase
- substractBonus
- validateUserPIN

Ανιχνευτές ορθής σειράς κλήσης των μεθόδων του API

Οι ανιχνευτές που χρησιμοποιήθηκαν για την ανίχνευση της ορθότητας της σειράς κλήσης των μεθόδων του API, ήταν δυο: ένας ο οποίος ανήκει στην οικογένεια visitor-based και ένας που χρησιμοποιεί τους *CFG*s. Ο λόγος που χρησιμοποιήθηκαν δυο διαφορετικοί ανιχνευτές, είναι ότι ο καθένας μπορεί να ανιχνεύσει διαφορετικές περιπτώσεις όπου γίνεται εσφαλμένη κλήση των επίφοβων μεθόδων.

Για να αποσαφηνιστεί ο σκοπός μας, δημιουργήσαμε παραδείγματα, όπου οι επίφοβες μέθοδοι είναι η *setOutgoing* και η *setOutgoingNoChaining*, της κλάσης APDU του API. Έτσι δημιουργήθηκε ένα αυτόματο, το οποίο έπρεπε να υλοποιηθεί για να ανακαλυφθούν τα σφάλματα μέσα στο δοκιμαζόμενο κώδικα. Το συγκεκριμένο αυτόματο φαίνεται στο παρακάτω σχήμα και μας πληροφορεί ότι σύμφωνα με τις προδιαγραφές των δυο επίφοβων μεθόδων, θα υπάρξει πρόβλημα όταν πραγματοποιηθέι διπλή κλήση της μιας ή της άλλης, ή κάποιος συνδυασμός αυτών.

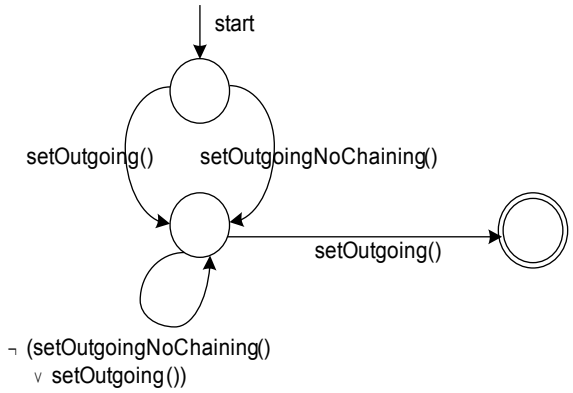
Περισσότερο επικεντρωθήκαμε στην μέθοδο setOutgoing, η οποία χρησιμοποιείται ώστε να δηλωθεί η κατεύθυνση της ροής των δεδομένων.

Συγκεκριμένα με αυτόν τον τρόπο ορίζουμε ότι θέλουμε να χρησιμοποιήσουμε την ροή της εξόδου, καθώς το πρόγραμμα επιθυμεί να στείλει δεδομένα με την μορφή πίνακα από byte. Η δομή μέσω της οποίας στέλνονται τα δεδομένα αυτά είναι ένα APDU πακέτο και ποιο συγκεκριμένα αποτελούν το μέρος της απάντησης, γνωστό και ως *ResponseAPDU*. Με την ενέργεια αυτή, κάθε δεδομένο στην είσοδο θα διαγραφεί και το πεδίο της κατάστασης του

εισερχόμενου *APDU* πακέτου θα τεθεί σε κατάσταση *STATE_OUTGOING*. Τέλος η επιστρεφόμενη τιμή μετά την κλήση της μεθόδου αντικατοπτρίζει το μέγιστο αναμενόμενο πλήθος των bytes που θα περιέχει η απάντηση.

Σύμφωνα με το API, η εξαιρέση που είναι πιθανών να εμφανιστεί μετά από λάθος κλήση της είναι η *APDUException*, με τους παρακάτω κώδικες σφάλματος:

- *APDUException.ILLEGAL_USE* εάν η μέθοδος αυτή ή η *setOutgoingNoChaining()* έχουν ήδη κληθεί
- *APDUException.IO_ERROR* για σφάλμα εισόδου / εξόδου (I/O error).



Η επόμενη εργασία, μετά από τον προσδιορισμό του αυτομάτου, ήταν να ανακαλυφθούν οι τρόποι με τους οποίους θα υλοποιηθεί αυτό, δηλαδή οι περιπτώσεις που υπάρχει πρόβλημα με την χρονική σειρά των κλήσεων των επίφοβων μεθόδων και που το αυτόματο φτάνει στην τελική κατάσταση του. Η πιο απλή περίπτωση είναι να υπάρχει διπλή κλήση της επίφοβης μεθόδου μέσα στο σώμα της ίδιας μεθόδου. Άλλη περίπτωση, είναι να καλείται η επίφοβη μέθοδος μέσα σε έναν βρόγχο και συνεπώς να υπάρχει πολλαπλή κλήση της. Μια πιο πολύπλοκη περίπτωση είναι να γίνεται αλυσιδωτή κλήση μεθόδων της εφαρμογής, από τις οποίες τουλάχιστον οι δύο περιέχουν κλήση μιας επίφοβης μεθόδου. Επιπλέον, μια ακόμη πιο πολύπλοκη περίπτωση είναι ο συνδυασμός των τελευταίων δυο περιπτώσεων, όπως η κλήση μίας μεθόδου που περιέχει κλήση μίας επίφοβης μεθόδου μέσα σε βρόγχο.

Ο visitor based ανιχνευτής

Ο πρώτος ανιχνευτής, που είναι και ο πιο απλός, κάνει μόνο ανίχνευση της πρώτης περίπτωσης. Έτσι με μια απλή γραμμική σάρωση του κώδικα, είναι εύκολο να γίνει αντιληπτό αν μία μέθοδος περιέχει διπλή κλήση της επίφοβης μεθόδου. Η υλοποίηση του αυτομάτου γίνεται με έναν απλό τρόπο. Πρώτα από όλα, υπάρχει μία μεταβλητή η οποία αναπαριστά το στάδιο που βρίσκεται το αυτόματο. Κάθε φορά που ο ανιχνευτής επισκέπτεται μία νέα μέθοδο, καλείται αυτόματα μια συγκεκριμένη μέθοδος, που εργασία της είναι να κάνει κάποια δουλειά όταν γίνεται είσοδος σε νέα μέθοδος, και η μεταβλητή που αναπαριστά το στάδιο λαμβάνει την τιμή που αντιπροσωπεύει την αρχική κατάσταση. Έτσι, όταν αρχίσει η αναζήτηση για την επίφοβη μέθοδο, το αυτόματο βρίσκεται στην αρχική κατάσταση και κάθε μια εντολή διοχετεύεται σε μία μέθοδο ώστε να αναλυθεί. Όταν βρεθεί η πρώτη κλήση της επίφοβης μεθόδου, αυτό σημαίνει ότι η μεταβλητή κατάστασης θα λάβει την τιμή της επόμενης κατάστασης. Αν ξαναβρεθεί η μέθοδος, τότε το αυτόματο φτάνει στην τελική κατάσταση και αναφέρει ένα σφάλμα. Συνοψίζοντας, όλα τα παραπάνω λειτουργούν με μία μεταβλητή που είναι ενδεικτική της κατάστασης και μία δομή, που όταν δει κλήση μίας επίφοβης μεθόδου, λαμβάνοντας υπόψιν την τρέχουσα κατάσταση ενεργεί κατάλληλα.

Ο CFG based ανιχνευτής

Ο δεύτερος ανιχνευτής είναι αρκετά πιο πολύπλοκος και έχει την ικανότητα να αναφέρει παραπάνω από μία περιπτώσεις, σφάλματος. Όπως είπαμε, αυτός ο ανιχνευτής βασίζεται στους CFGs και χρησιμοποιεί τα χαρακτηριστικά τους για να κάνει ανίχνευση των εσφαλμένων περιπτώσεων χρήσης των επίφοβων μεθόδων.

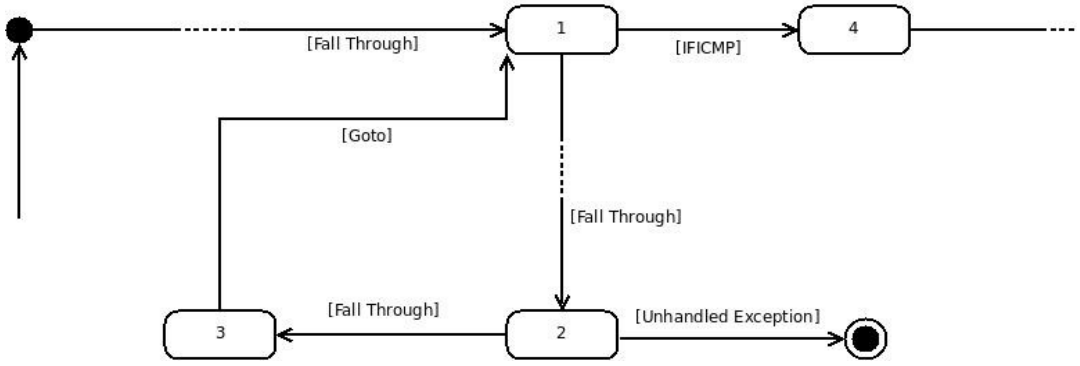
Σε αυτό το σημείο είναι σημαντικό να αναφέρουμε τι είναι ένας γράφος ροής ελέγχου (*Control Flow Graph*) καθώς και πώς μπορούμε να χρησιμοποιήσουμε τα χαρακτηριστικά του. Ο CFG που κατασκευάζει το FindBugs, είναι ο γράφος ροής ελέγχου μιας μεθόδου και αναπαριστά την ροή ελέγχου του προγράμματος. Τα δομικά στοιχεία ενός CFG, είναι οι ακμές και οι κόμβοι, όπως και σε κάθε γράφο. Οι κόμβοι αναπαριστούν τα *μπλοκ κώδικα* (*Basic Blocks*), ενώ οι *ακμές* (*Edges*) συνδέουν τα μπλοκ αυτά, σύμφωνα με την ροή του προγράμματος. Από τα μπλοκ κώδικα είναι δυνατή η ανάκτηση του κώδικα, ώστε να γίνει πιο λεπτομερής έλεγχος σε επίπεδο κώδικα. Οι

ακμές μπορούν να έχουν διαφορετικούς τύπους, ανάλογα με την σύνδεση που πραγματοποιούν, μέσα στο FinddBugs. Μερικοί τύποι των ακμών είναι:

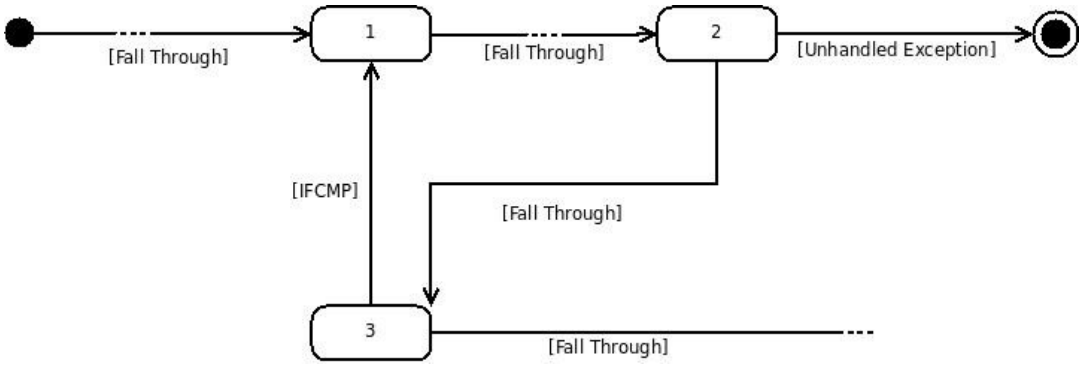
- *Fall Through* – Απλό πέρασμα από ένα μπλοκ στο επόμενο κατά σειρά μπλοκ
- *IFCMP* – Πέρασμα στο επόμενο μπλοκ μετά από σύγκριση
- *Goto* – Πέρασμα σε ένα μπλοκ το οποίο δεν βρίσκεται σε σειρά
- *Unhandled Exception* – Πέρασμα ροής σε ειδικό μπλοκ, αφού υπάρξει αδιαχείριστη εξαίρεση

Ο CFG μιας μεθόδου, είναι προσβάσιμος από τον ανιχνευτή από μια ειδική δομή που περιέχει πληροφορίες για την εφαρμογή που βρίσκεται υπό ανάλυση, την *classContext*. Η κατασκευή του απαιτεί σχετικά αρκετό χρόνο, όπως και η αναζήτηση μέσα σε αυτόν.

Πρώτα, θα αναφερθούμε στην ανίχνευση των κλήσεων επίφοβων μεθόδων που βρίσκονται μέσα σε βρόγχους. Αυτό γίνεται με την επίσκεψη του κάθε μπλοκ και την εποπτεία των ακμών του. Έτσι, ένα μπλοκ το οποίο έχει δυο εισερχόμενες ακμές, μία Fall *Through* και μία *Goto*,και μια εξερχόμενη IFCMP, είναι γνωστό ότι είναι ένα μπλοκ απο το οποίο ξεκινά ένας βρόγχος *for* ή *while*. Συνεπώς, η αναζήτηση για κλήση της επίφοβης μεθόδου θα γίνει στα μπλοκ που ακολουθούν, όπου το πρώτο από αυτά είναι εκείνο στο οποίο εισέρχεται μία *FallThrough* ακμή που ξεκινάει από το μπλοκ που είπαμε πιο πάνω και τα υπόλοιπα είναι εκείνα τα οποία είναι προσπελάσιμα απο μία *Fall Through* ακμή. Το κομμάτι του γράφου που αντιστοιχεί στην δομή ενός βρόγχου απεικονίζεται παρακάτω, με τον κόμβο – κλειδί από όπου αρχίζει ο βρόγχος να είναι αυτός με τον αριθμό ένα και οι κόμβοι που περιέχουν τον κώδικα του βρόγχου με τους αριθμούς δύο και 3. Ο κόμβος με τον αριθμό τέσσερα είναι το μπλοκ που ακολουθεί τον βρόγχο.



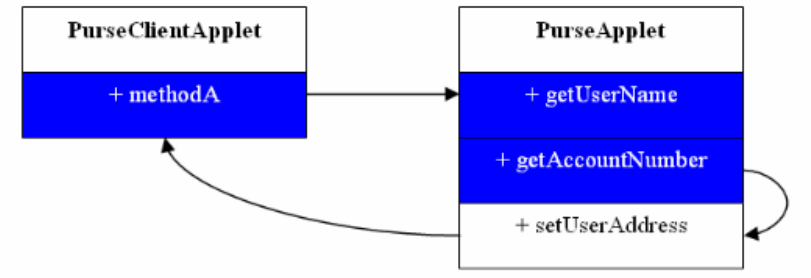
Εκτός όμως από τους κλασσικούς βρόγχους *for* και *while*, υπάρχει και η δομή βρόγχου *do – while*, η οποία εμφανίζεται διαφορετικά στον ενδιάμεσο κώδικα της Java, και συνεπώς πρέπει να γίνει διαφορετικός αλγόριθμος, με την ίδια όμως λογική με τον παραπάνω, ο οποίος θα ανιχνεύει τον βρόγχο. Εκτός αυτού, η αναζήτηση στα μπλοκ που βρίσκονται μέσα στο βρόγχο είναι η ίδια με προηγουμένως. Έτσι στο παρακάτω σχήμα ο κόμβος που ξεκινά τον βρόγχο είναι αυτός με τον αριθμό ένα και οι ενδιάμεσοι με τους αριθμούς δύο και τρία.



Επίσης στον ίδιο ανιχνευτή υπάρχει κώδικας ο οποίος ανιχνεύει την περίπτωση που υπάρχει αλυσιδωτή κλήση μεθόδων, που περιέχουν κλήση της επίφοβης μεθόδου. Η ανίχνευση αυτών των σφαλμάτων απαιτεί κάποια προεπεξεργασία. Η προεπεξεργασία περιλαμβάνει διάφορα στάδια, μέχρι να εκμαιεύσουμε τα δεδομένα που χρειάζονται ώστε να πραγματοποιηθεί η ανάλυση. Πρώτα, μέσω μίας κλάσης που έχει υλοποιηθεί, κατασκευάζεται ο γράφος κλήσης όλης της εφαρμογής που αναλύεται. Αυτός περιέχει τις κλήσεις που πραγματοποιούνται μέσα στην εφαρμογή και αναπαριστά τις μεθόδους με

κόμβους και οι ακμές συμβολίζουν την κλήση που πραγματοποιείται. Έπειτα, γίνεται μία διάσχιση όλων των μεθόδων, από την οποία θα λάβουμε σαν αποτέλεσμα μία δομή που θα περιέχει τις μεθόδους της εφαρμογής που καλούν την επίφοβη μέθοδο. Αυτή ή δομή ονομάζεται *"μαύρη λίστα"*. Τέλος, κάνουμε διάσχιση του γράφου ξεκινώντας από κάθε κόμβο – μέθοδο, ώστε να γίνει αντιληπτό αν μία κλήση που ξεκινάει από μία δεδομένη μέθοδο, εμπεριέχει (έστω και μετά από αλυσιδωτή κλήση) κλήση μίας μεθόδου που βρίσκεται στην *"μαύρη λίστα"*. Οι μέθοδοι που ικανοποιούν την παραπάνω συνθήκη τοποθετούνται σε μία δομή που ονομάζεται *"γκρι λίστα"*.

Αφού είναι έτοιμα τα δεδομένα για να γίνει η περαιτέρω ανάλυση, ξεκινάει μία διάσχιση του γράφου από κάθε κόμβο – μέθοδο, σύμφωνα με τις αλυσιδωτές κλήσεις που γίνονται, και κατά την διάρκεια της διάσχισης ελέγχουμε τις μεθόδους που καλούνται αν βρίσκονται στην μαύρη ή στην γκρι λίστα. Έτσι είναι δυνατή η εύρεση διπλών κλήσεων της επίφοβης μεθόδου, έστω και αν βρίσκονται σε διαφορετικές μεθόδους. Επίσης, ελέγχεται ταυτόχρονα με την κλήση της επίφοβης μεθόδου μέσα σε ένα βρόγχο αν μέσα στον βρόγχο καλείται μία μέθοδος της γκρι ή μαύρης λίστας, διότι θα υπάρξει πρόβλημα. Το παρακάτω σχήμα παριστάνει την κλήση αλυσιδωτών μεθόδων, όπου οι μπλε μέθοδοι περιέχουν κλήση της επίφοβης μεθόδου.



Η ανάλυση που επιτελούν οι παραπάνω ανιχνευτές, είναι ανεξάρτητη της ροής του προγράμματος (*path insensitive*), καθώς δεν λαμβάνουν υπόψιν τους τα εναλλακτικά μονοπάτια που μπορούν να ακολουθηθούν. Έτσι, αν για παράδειγμα υπάρχει μια δομή if else στον κώδικα μας και μέσα στην *if* όπως και στην *else* υπάρχει κλήση μιας επίφοβης μεθόδου, είναι κατανοητό ότι δεν μπορεί να υπάρξει πρόβλημα, διότι δεν είναι δυνατόν

να εκτελεστούν και τα δυο μπλοκ κώδικα σε μια εκτέλεση της μεθόδου. Παρόλα αυτά, οι συγκεκριμένοι ανιχνευτές εντοπίζουν τον κώδικα αυτό σαν προβληματικό και αναφέρουν το σφάλμα. Εδώ είναι η περίπτωση που είχε αναφερθεί, ότι δηλαδή είναι καλύτερο μερικές φορές να έχουμε εσφαλμένα αποτελέσματα, όσο αφορά εύρεση σφαλμάτων που δεν υπάρχουν, γιατί η ολοκλήρωση του κώδικα απαιτεί αρκετό κόπο.

Ανιχνευτές ορθότητας ορισμάτων του API

Ένας άλλος ανιχνευτής που δημιουργήθηκε ήταν αυτός που έχει την δυνατότητα ανίχνευσης σφαλμάτων που έχουν σχέση με τις τιμές που ανατίθενται στις διάφορες μεταβλητές κατά την διάρκεια εκτέλεσης του προγράμματος, οι οποίες θα χρησιμοποιηθούν σαν ορίσματα για κάποια μέθοδο του API. Αυτά τα σφάλματα εξαρτώνται από την ροή της εκτέλεσης (*Data Flow*) του προγράμματος και είναι δύσκολο να προσδιοριστούν όταν υπάρχουν πολλά μονοπάτια στο πρόγραμμα ή όταν η τιμή μίας μεταβλητής εξαρτάται από εξωτερικούς παράγοντες, όπως από την είσοδο του χρήστη. Είναι σημαντικό να τονίσουμε ότι το FindBugs παρέχει κάποια βασικά στοιχεία ώστε να γίνει μία υποτυπώδης ανάλυση. Σε καμία περίπτωση όμως δεν μπορεί να θεωρηθεί ολοκληρωμένη αυτή η ανάλυση, διότι ολόκληρη η ανάλυση ροής δεδομένων που πραγματοποιείται έχει εμβέλεια μίας μεθόδου.

Η τεχνική αυτή βασίζεται στους *CFG*, καθώς αξιοποιούν τα μπλοκ κώδικα και τις ακμές ώστε να προσομοιώσουν την ροή που θα υπήρχε σε ένα εκτελούμενο πρόγραμμα. Βασική ιδέα στην ανάλυση ροής δεδομένων είναι τα γεγονότα (*facts*), τα οποία περιγράφουν την κατάσταση ορισμένων μεταβλητών μέσα στον *CFG*. Αυτά δημιουργούνται και λαμβάνουν τιμές ανάλογα με την ροή του προγράμματος και τις εντολές που επιτελούνται. Πριν από κάθε *μπλοκ κώδικα*, το οποίο είναι ένας *κόμβος* του *CFG*, υπάρχει κάποιο γεγονός το οποίο θα αποτελέσει δεδομένο εισόδου ώστε να υπολογιστεί το γεγονός μετά από το μπλοκ. Όταν δυο μονοπάτια συναντώνται τότε τα γεγονότα μας συγχωνεύονται. Ο υπολογισμός, κατασκευή και διαχείριση των γεγονότων περιγράφεται από την θεωρία πλέγματος (*lattice theory*), η ανάλυση της οποίας είναι πέρα από τα όρια της παρούσας εργασίας.

Στο FindBugs, οι ανάλυσεις ροών δεδομένων, υλοποιούνται από κατάλληλες κλάσεις προσφέροντας μια μικρή γκάμα επιλογών. Μέσα στην γκάμα αυτή μπορούμε να βρούμε αναλύσεις που μπορούν να εποπτεύσουν την τιμή κάποιας μεταβλητής, το αν είναι κενή (null), τον τύπο της, αλλά και την προέλευση της όπως αν για παράδειγμα είναι επιστρεφόμενη τιμή. Όπως είπαμε προηγουμένως, η γκάμα εκτός από μικρή δεν είναι τελειοποιημένη και προσφέρει βασικές λειτουργίες με ανάλυση εμβέλειας μεθόδου.

Παρόλα όμως τα μειονεκτήματα αυτά, με τον συνδυασμό ορισμένων αναλύσεων είναι δυνατή η εποπτεία ορισμένων μεταβλητών. Συγκεκριμένα, με την χρήση της ανάλυσης που βασίζεται στα πλαίσια (*FrameDataflowAnalysis*), η οποία αναπαριστά τις μεταβλητές και τις τιμές που παίρνει η στοίβα τελεστών, της ανάλυσης ελέγχου κενής τιμής (*IsNullValueAnalysis*), της ανάλυσης που δίνει πληροφορίες σχετικά με τον τύπο των μεταβλητών (*ContantAnalysis*) και της ανάλυσης που εποπτεύει την παραγωγή και την ροή των τιμών στην στοίβα της Java (*ValueNumberAnalysis*), ήταν δυνατός ο έλεγχος των ορισμάτων της μεθόδου:

```
short arrayCopy(byte[] src, short srcOff, byte[] dest,
                short destOff, short length)
```

Αυτή, όπως βλέπουμε δέχεται πέντε ορίσματα και απαιτεί τα δυο από αυτά να έχουν μη κενή τιμή (πρώτο και τρίτο) και τα άλλα τρία θετική (δεύτερο, τέταρτο και πέμπτο). Έτσι μέσα από τον ανιχνευτή, όταν υπάρξει μια κλήση σε αυτή την μέθοδο, πραγματοποιούμε όλες τις παραπάνω αναλύσεις για κάθε όρισμα της και αξιοποιούμε τα στοιχεία που χρειαζόμαστε από κάθε ανάλυση, ώστε να καταλάβουμε αν πληρούνται οι προϋποθέσεις σε κάθε ένα από αυτά. Πολλές φορές, δεν είναι δυνατόν να εξαχθεί κάποιο συμπέρασμα, λόγω παραγόντων εξωτερικά της μεθόδου προς ανάλυση, καθιστώντας την ανάλυση ασαφή. Ο παρακάτω ψευδοκώδικας περιγράφει την εργασία του ανιχνευτή.

for each method in the class **do**

request a CFG for the method
get the method's ConstantDataflow from ClassContext
get the method's ValueNumberDataflow from ClassContext
get the method's IsNullValueDataflow from ClassContext
for each location in the method **do**
get instruction handle from location
get instruction from instruction handle

```
            if instruction is not instance of invoke static then
                continue
            end if
            get the invoked method's name from instruction
            get the invoked method's signature from instruction
            if invoked method is arrayCopy then
                get ConstantFrame (fact) at current location
                get ValueNumberFrame (fact) at current location
                get IsNullValueFrame (fact) at current location
                get the method's number of arguments
                    for each argument do
                        get argument as
                            Constant, ValueNumber,IsNullValue
                        if argument is constant then
                            if argument is negative then
                                report a bug
                            end if
                        else
                        if argument is not method return value
                        nor constant then
                            if argument is not definitely not null then
                                report a bug
                            end if
                        end if
                    end for
            end if
        end for
end for
```

Επίλογος

Η εργασία αυτή εξερεύνησε την επάρκεια του εργαλείου FindBugs, πάνω στον έλεγχο προγραμμάτων έξυπνων καρτών με στατική ανάλυση προγράμματος. Τελικά με την υλοποίηση δύο ομάδων ανιχνευτών, έγινε δυνατή η χρησιμοποίηση των χαρακτηριστικών του. Με την συγγραφή του κατάλληλου κώδικα δημιουργήθηκε ένα πρόσθετο το οποίο είναι ικανό να ελέγξει ορισμένες από τις περιπτώσεις όπου η ασφάλεια είναι πιθανόν να παραβιασθεί. Γενικεύοντας το, είναι δυνατόν να δημιουργηθούν αρκετοί ανιχνευτές, που θα ελέγχουν και για άλλα χαρακτηριστικά, με αποτέλεσμα να έχουμε ένα ολοκληρωμένο σετ ανιχνευτών που θα καταστήσει την εργασία των προγραμματιστών πιο απλή και λιγότερο χρονοβόρα. Μελλοντικά θα ήταν

χρήσιμη η συγγραφή κώδικα μέσα στο FindBugs, ο οποίος θα έχει την δυνατότητα της ανάλυσης πέρα από τα στενά όρια της μιας μεθόδου.

Περιεχόμενα του CD

Στο CD περιέχεται το project στον φάκελο με όνομα κώδικας ο οποίο περιέχει τον πηγαίο κώδικα των εφαρμογών που αναπτύχθηκαν:

- APDU εφαρμογές

  Ο φάκελος αυτός περιέχει την εφαρμογή που αναπτύχθηκε με τεχνολογία APDU. Η εφαρμογή αυτή έχει δύο εκδόσεις όπου η μία προορίζεται για εκτέλεση στην κάρτα και η άλλη για εκτέλεση με το εργαλείο προσομοίωσης cref. Σε κάθε έκδοση υπάρχουν οι αντίστοιχοι φάκελοι που διαχωρίζουν τον κώδικα κάρτας, τα applets, από τις εφαρμογές πελάτη, σε Java.
- RMI εφαρμογή

  Στο φάκελο src της RMI εφαρμογής μπορείτε να βρείτε τον πηγαίο κώδικα. Συγκεκριμένα στο φάκελο purseclient βρίσκεται η εφαρμογή εκτός κάρτας ενώ στο φάκελο RMIDemo ο κώδικας που φορτώνεται στην κάρτα.

Ακόμα στο φάκελο APDU εφαρμογές βρίσκεται ο φάκελος `BatchFiles` που περιέχει αρχεία δέσμης για την αυτοματοποίηση των διεργασιών. Για να πραγματοποιήσουμε όλη την παραπάνω διαδικασία πρέπει, εκτός από το να εγκαταστήσουμε όλα τα development kit, να τοποθετήσουμε τον φάκελο τον αντίστοιχο φάκελο στην διαδρομή `C:\`. Όταν γίνουν όλα αυτά είμαστε έτοιμοι να εκτελέσουμε όλες τις παραπάνω εντολές. Η εφαρμογή τρέχει σε περιβάλλον Windows XP.

Σχετικά με την στατική ανάλυση προγραμμάτων έξυπνων καρτών, περιλαμβάνεται το FindBugs στον κατάλογο `Findbugs`, το πρόσθετο που περιέχει τους ανιχνευτές των χαρακτηριστικών στον φάκελο `JavaCardPlugin`, καθώς και περιπτώσεις με προγράμματα έξυπνων καρτών για έλεγχο του πρόσθετου, στο `TestCases`.

Αναφορές

1. Zhiqun Chen, Java Card(TM) Technology for Smart Cards: Architecture and Programmer's Guide, pp. 12-18, 29-32, 105-108

2. Enrique Ortiz, An Introduction to Java Card Technology - Part 3, The Smart Card Host Application, September 2003

3. Ed Ort, Developing a Java Card Applet, Release 2.1.2, August 2001

4. Ed Ort. Writing a Java Card Applet, Release 2.1.1, January 2001

5. Wikipedia, Smart Cards, http://java.sun.com/products/javacard/smartcards.html, 5 March 2008

6. Vesna Hassler, Martin Manninger, Mikhail Gordeev, Christoph Muller, Java Card for E-Payment Applications, 2002

7. CardWerk website, Smart Card Operating System, March 1, 2008

8. Smartcardbasic website, Types of Cards ,www.smartcardbasics.com, 2005

9. Jan De Clercq, Microsoft website: http://www.microsoft.com/technet/security/guidance/identitymanagement/scard.mspx, Smart Cards

10. Rinaldo Di Giorgio, JavaWorld.com: Smart cards: A primer Develop on the Java platform of the future, 12/01/97

11. Zhiqun Chen, JavaWorld.com: How to write a Java Card applet: A developer's guide, Learn the programming concepts and major steps of creating Java Card applets, 07/01/99

12. Zhiqun Chen and Rinaldo Di Giorgio, JavaWorld.com: Understanding Java Card 2.0, Learn the inner workings of the Java Card architecture, API, and runtime environment, 03/01/98