TRACER

Κωδικός Έργου: 09ΣΥΝ-72-942

Π2.3 - Σχεδιασμός περιβάλλοντος υποστήριξης ανάπτυξης της πλατφόρμας

Παραδοτέο έργου

Ενότητα Εργασίας:	2: Work Package Title
Αριθμός Παραδοτέου:	2.3
Συντονιστής:	Σ. Ιωαννίδης (Ι.Τ.Ε.)
Συντελεστές:	Contributors
Ημερομηνία υποβολής:	1 Ιουνίου 2012
Ημερομηνία παράδοσης:	30 Οκτωβρίου 2012
Αναγνωριστικό εγγράφου:	TRACER_Π_2.3

Περιεχόμενα

1 Εισαγωγή

Σε αυτό το παράδοτέο περιγράφεται ο σχεδιασμός του περιβάλλοντος υποστήριξης της πλατφόρμας TRACER.

1.1 Σκοπός

Σκοπός του παραδοτέου είναι η λεπτομερής σχεδίαση του περιβάλλοντος υποστήριξης και ανάπτυξης της πλατφόρμας TRACER, ώστε να μπορεί να επεκταθεί με την χρήση εργαλείων στατικής ανάλυσης κώδικα μέσω λειτουργικών αρθρωμάτων.

1.2 Σύνοψη

Για την θωράκιση ασφάλειας των συστημάτων παλαιάς γενιάς είδαμε, από το παραδοτέο 1.1 (Επισκόπηση του State of the Art), πως έχει αναπτυχθεί μια πλειάδα εργαλείων και μεθόδων ανάλυσης. Προκειμένου να εκμεταλλευτούμε τις δυνατότητες που προσφέρουν τα παραπάνω προγράμματα και μέθοδοι στην πλατφόρμα TRACER είναι αναγκαία η σχεδίαση μιας προγραμματιστικής διεπαφής **API!** (**API!**) που θα είναι υπεύθυνη για την αλληλεπίδραση των εργαλείων στατικής ανάλυσης και της πλατφόρμας TRACER.

Το προγραμματιστικό παράδειγμα με χρήση αρθρωμάτων (modular programming paradigm) χρησιμοποιείται κατά κόρον από συστήματα και εφαρμογές στοχεύοντας στην προσθήκη λειτουργικότητας. Βασισμένο στην τεχνική "διαίρει και βασίλευε", ένα σύστημα αποτελείται από μικρά, αυτοτελή κομμάτια κώδικα που επιτελούν συγκεκριμένες λειτουργίες. Τα κομμάτια αυτά, τα οποία ονομάζονται αρθρώματα (plugins) επικοινωνούν μέσω μιας προγραμματιστικής διεπαφής - **API!** - με τον πυρήνα του συστήματος. Χαρακτηριστικά είναι τα παραδείγματα του Firefox [1], των εργαλείων ανάπτυξης Eclipse [4] και Netbeans [3] αλλά και της πλατφόρμας ηλεκτρονικής μάθησης Moodle [2] που χρησιμοποιούν αρθρώματα για την επέκταση της λειτουργικότητας τους. Τα οφέλη από αυτήν την σχεδίαση αντικατοπτρίζονται τόσο στην υλοποίηση του ίδιου του συστήματος (επαναχρησιμοποίηση κώδικα, δυνατότητα υλοποίησης διαφορετικών λειτουργιών ταυτόχρονα) όσο και στην επεκτασιμότητα του (νέα χαρακτηριστικά μπορούν να προστεθούν αργότερα).

Με αυτό τον τρόπο, η υπάρχουσα λειτουργικότητα από μεθόδους και εργαλεία ανάλυσης θα μπορεί να ενσωματωθεί στην πλατφόρμα TRACER με την χρήση λειτουργικών αρθρώματων (plugins), τα οποία επικοινωνούν με τον πυρήνα της πλατφόρμας μέσω μιας καλά ορισμένης προγραμματιστικής διεπαφής. Μέσω αυτής, τα διάφορα αρθρώματα θα επιτελούν καθορισμένες λειτουργίες όπως πχ την ανίχνευση και καταγραφή συγκεκριμένων τρωτοτήτων στον κώδικα του υπό εξέταση συστήματος παλαιάς γενιάς. Για την ευκολότερη ενσωμάτωση των εργαλείων η πλατφόρμα θα παρέχει στον προγραμματιστή των αρθρωμάτων τη δυνατότητα δημιουργίας ενός αρχέτυπου του αρθρώματος. Το αρχέτυπο θα περιέχει κενές μεθόδους, τις οποίες ο προγραμματιστής καλείται να υλοποιήσει.

1.3 Δομή

Η διάρθρωση του παραδοτέου είναι η ακόλουθη: Στην ενότητα 2, παρουσιάζονται τα διαθέσιμα εργαλεία στατικής ανάλυσης. Οι λειτουργίες που πρέπει να υποστηρίζονται από την πλατφόρμα και από τα λειτουργικά της αρθρώματα περιγράφονται στην ενότητα 3, ενώ στην ενότητα 4 γίνεται λεπτομερής ορισμός των διεπαφών που θα χρησιμοποιηθούν στην πλατφόρμα. Τα συμπεράσματα του παραδοτέου δίνονται στην ενότητα 6.

2 Κατηγορίες Εργαλείων

(συγγραφή: AUTH)

2.1 Υπάρχοντα εργαλεία

FindBugs

Το FindBugs [9, 7] είναι ένα ανοιχτού κώδικα εργαλείο στατικής ανάλυσης που ψάχνει για τρωτότητες. Εξετάζει τα μεταγλωττισμένα αρχεία Java των προγραμμάτων προς έλεγχο, χρησιμοποιώντας την bytecode engineering library (BCEL) [8]. Υποστηρίζει την ενσωμάτωση ανιχνευτών τρωτοτήτων ως plug-ins και περιλαμβάνει ένα επεκτάσιμο μηχανισμό αναφοράς λαθών, τόσο μέσω ενός GUI όσο και μέσω αναπαράστασης κειμένου. Για την ανίχνευση μιας τρωτότητας, το FindBugs χρησιμοποιεί διάφορες μεθόδους. Για παράδειγμα, για ανιχνεύσει τρωτότητες τύπου null pointer, το FindBugs χρησιμοποιεί ανάλυση ροής ελέγχου (control flow analysis) και ανάλυση ροής δεδομένων (data flow analysis). Επίσης έχει και άλλους ανιχνευτές που χρησιμοποιούν visitor patterns στις κλάσεις και στις μεθόδους, χρησιμοποιώντας μηχανές καταστάσεων (state machines) για να αιτιολογήσουν την ύπαρξη συγκεκριμένων τιμών στις μεταβλητές ή στην στοίβα (stack) του προγράμματος. Οι προειδοποιήσεις που εμφανίζει το FindBugs κατηγοριοποιούνται αναλογα τον τύπο της τρωτότητας και άρα σε κατηγορίες όπως correctness, malicious code vulnerability και bad practice. Εδώ, ενδιαφερόμαστε μόνο για δύο κατηγορίες: Για τις security και τις malicious code vulnerability προειδοποιήσεις.

Το FindBugs έχει χρησιμοποιηθεί πάρα πολλές φορές τόσο για επιστημονικές όσο και για εμπορικές ανάγκες. Για παράδειγμα, χρησιμοποιήθηκε για να αναλύσει όλα τα διαθέσιμα builds του **JDK!** (**JDK!**) [6], ενώ η Google το έχει ενσωματώσει στην διαδικασία παραγωγής λογισμικού της [5, 10]. Επίσης, έχει επεκταθεί ωστέ να επαληθεύει κλήσεις προγραμματιστικών διεπαφών (**API!**) [11] και να ανακαλύπτει τρωτότητες σε AspectJ εφαρμογές [10].

Προκειμένου να εξεταστεί ο δυαδικός κώδικας του έργου χωρίς τις εξαρτήσεις του, συλλέγονται όλα τα αντίστοιχα πακέτα κλάσεων και χρησιμοποιείται η επιλογή `-only Analyze` του FindBugs για να τα εισάγουμε ως μια παράμετρο. Χρησιμοποιώντας την επιλογή `-xml`, η αναφορά θα περιέχει τις περιγραφές των τρωτοτήτων σε μορφή **XML!** (**XML!**). Συνεπώς είναι εύκολο να αναζητήσουμε μέσα στην αναφορά τις τρωτότητες, οι οποίες μας ενδιαφέρουν. Οι τρωτότητες συσχετίζονται με πληροφορίες της συγκεκριμένης έκδοσης του αρχείου, οι οποίες είναι αποθηκευμένες μέσα στην βάση δεδομένων.

FRAMA-C

Το FRAMA-C είναι μια πλατφόρμα, η οποία ασχολείται με την στατική ανάλυση πηγαίου κώδικα της γλώσσας προγραμματισμού C. Η πλατφόρμα συγκεντρώνει πολλές τεχνικές στατικής ανάλυσης σε ένα συνεργατικό επεκτάσιμο πλαίσιο. Η συνεργατική προσέγγιση του FRAMA-C επιτρέπει σε εργαλεία στατιής αναλυσης να μεταγλωττίσουν πάνω σε ήδη υπολογισμένα αποτελέσματα από άλλα εργαλεία μέσα σε αυτό το πλαίσιο. Συνεπώς, το FRAMA-C μπορεί να προσφέρει έναν αριθμό από εξεζητημένα εργαλεία αναλύσεων (με την μορφή plugins) όπως:

value analysis υπολογίζει μια τιμή ή ένα σύνολο πιθανών τιμών για κάθε μεταβλητή του προγράμαμτος. Αυτό το plug-in χρησιμοποιείεται και από άλλα plug-in.

jessie επιβεβαιώνει ιδιότητες με έναν επαγωγικό τρόπο.

impact analysis τονίζει μέσα στον πηγαίο κώδικα της C τις επιπτώσεις μιας αλλαγής.

slicing "τεμαχίζει" το πρόγραμμα. Δίνει την δυνατότητα να παραχθεί ένα νέο πρόγραμμα C, το οποίο διατηρεί κάποιες από τις ιδιότητες του αρχικού.

dependency analyzer δίνει την δυνατότητα για ανάλυση των εξαρτήσεων σε ένα πρόγραμμα C.

Το FRAMA-C είναι παρόμοιο με κάποιο εργαλεία ευρηστικής ανίχνευσης κενών ασφάλειας, αλλά διαφέρει σε δύο σημαντικά σημεία με αυτά:

- στοχεύει να είναι σωστό, το οποίο σημαίνει να μην αποκρύπτει κανένα σημείο του κώδικα, στο οποίο μπορεί να δημιουργήθει σφάλμα κατά την εκτέλεση.
- επιτρέπει στον χρήστη να καθορίζει τις λειτουργικές προδιαγραφές και να αποδεικνύει οτι ο κώδικας τις ικανοποιεί.

Το FRAMA-C μπορεί να εγγυηθεί ότι δεν υπάρχουν κενά ασφάλειας σε ένα πρόγραμμα το οποίο σημαίνει είτε σφάλμα κατά την ετκέλεση είτε κάποια απόκλιση από την καθορισμένη λειτουργική προδιαγραφή. Αυτό βεβαία απαιτεί από τον χρήστη περισσότερη δουλειά σε σχέση με άλλα εργαλεία. Επίσης, επειδή μερικά εργαλεία επιτρέπουν σχετικά μικρή επέμβαση στις ρυθμίσεις, την δυνατότητα αυτή την δίνει η συνεργατική προσέγγιση του FRAMA-C όσον αφορά τα plug-in του.

Τα εργαλεία του FRAMA-C μπορούν να βοηθήσουν στην καλύτερη κατανοήση καλύτερα του πηγαίου κώδικα από τον χρήστη, επειδή δίνουν την δυνατότητα να εξάγουν πληροφορίες από αυτόν. Αρκετά plug-in του FRAMA-C μπορουν να αποκαλύψουν τι ακριβώς κάνει ένα πρόγραμμα C. Μερικά από αυτά τα plug-in μπορούν:

- να παρατηρήσουν ένα σύνολο πιθανών τιμών των μεταβλητών του προγραμμάτος σε κάθε σημείο της εκτέλεσης.
- να "τεμαχίσουν" το αρχικό πρόγραμμα σε μικρότερα.
- να κατευθύνουν την ροή δεδομένων από τον ορισμό στην χρήση ή και το αντίστροφο.

Μετά τον καθορισμό του έργου προς ανάλυση, καλείται το FRAMA-C. Καθορίζεται το plug-in, το οποίο θα κληθεί για να κάνει την ανάλυση. Για παράδειγμα, μπορεί να χρησιμοποιηθεί η `Value Analysis` με την επιλογή `-val`. Επίσης, για την `Taint Analysis` χρησιμοποιείται η επιλογή `-taint-analysis` δηλώνοντας το αρχείο ρυθμίσεων με την εντολή `-config-file` και το συμπληρωματικό αρχείο ρυθμίσεων με την εντολή `-constr-config -file`. Τέλος, καθορίζεται ο τρόπος με τον οποίο θα εμφανιστούν τα αποτελέσματα. Με την επιλογή `-ocode` ανακατευθύνεται τα αποτελέσματα του επιθυμητού έργου σε ένα αρχείο, και με την επιλογή `-print-final` εμφανίζονται τα αποτελέσματα του επιθυμητού έργου στην κονσόλα.

AMNESIA

Το AMNESIA κάνει στατική ανάλυση στον πηγαίο κώδικα της Java χρησιμοποιώντας ένα εργαλείο στατικής ανάλυσης το JavaStringAnalyzer. Αρχικά, θα πρέπει να καθοριστούν τα αρχεία πηγαίου κώδικα της Java τα οποία θα αναλυθούν. Στη συνέχεια, ξεκινάει η διαδικασία ανάλυσης με το AMNESIA, του οποίου τα βήματα είναι:

- Εύρεση εντολών **SQL!** (**SQL!**) (Identify hot spots), όπως η συνάρτηση

  ```
  java.sql.Statement.execute(String)
  ```

- Δημιουργία **SQL!** μοντέλων (Build SQL-Query models) με τη χρήση του Java String Analysis (JSA) και την αποθήκευση των μη ντετερμινιστικά πεπερασμένων αυτόματων, τα οποία παράγονται (Non-Deterministic Finite Automaton (NDFA)s). Ομαδοποίηση χαρακτήρων όπως λέξεις **SQL!**, τελεστές ή τιμές ως ένα κόμβο και δημιουργία μιας μετάβασης.
- Ενσωμάτωση μοντέλων στην εφαρμογή, όπως συμβαίνει στο κομμάτι κώδικα της Listing 1.
- Έλεγχος κατά την διάρκεια της εκτέλεσης.

Listing 1: Κώδικας ενσωμάτωσης Monitor στον πηγαίο κώδικα ενός προγράμματος

```
if (monitor.accepts (<hotspot ID>, queryString)) {
        ResultSet tempSet = stmt.execute(queryString);
        return tempSet;
}
```

WASP

Το WASP εξετάζει αρχεία πηγαίου κώδικα της Java εφαρμόζωντας μια δυναμική ανάλυση, η οποία βασίζεται στην τεχνική της θετικής tainted ανάλυσης. Αρχικά, θα πρέπει να καθοριστούν τα αρχεία πηγαίου κώδικα της Java τα οποία θα αναλυθούν. Στη συνέχεια, ξεκινάει η διαδικασία ανάλυσης με το WASP, του οποίου τα βήματα είναι:

- Χρήση μιας MetaStrings Specialized Library (assign and track trust marking at character level at runtime), η οποία επεκτείνει την συμπεριφορά των κλάσεων που χειρίζονται τα αλφαριθμητικά. Δεν κάνει να εφαρμοστεί στις παρακάτω περιπτώσεις (οι οποίες δεν συναντώνται σε **SQL!** ερωτήματα)
  - Primitive types (char)
  - Native methods (αντικείμενο της string ως παράμετρος σε μια native method)
  - Στα hard-coded strings δημιουργημένα από string constructor
- Ενσωμάτωση του positive tainting μέσω του String Initializer and Instrumenter και αφορά αυτά τα τρία είδη αλφαρηθμιτικών hard-coded, εμμέσως δημιουργημένα από την java, ως είσοδο από τρίτους).

- Ενσωμάτωση κλήσεων στην String Checker. Χρήση κάποιου parser της SQL για να δημιουργήσει tokens. Ελεχονται αυτά τα tokens με την προκαθορισμένη πολιτική, αν περιέχουν κάποιον χαρακτήρα που δεν πρέπει. Μπορούν να χρησιμοποιηθούν και άλλες πολιτικές.
- Τέλος, υπάρχει η επιλογή της διαχείρισης των “False positive” με την διαδικασία τουτ “learning mode”, η οποία δείχνει στον προγραμματιστή που έγινε το σφάλμα.

Soot

Το Soot είναι ένα πλαίσιο βελτιστοποίησης των προγραμμάτων σε Java. Μπορεί να εκτελέσει ένα μεγάλο σύνολο αναλύσεων στατικής ανάλυσης σε προγράμματα Java είτε σε αρχεία πηγαίου κώδικα είτε σε δυαδικά αρχεία. Ένα από τα πλεονεκτήματα του Soot είναι ότι μπορεί να δημιουργήσει αυτόματα τέσσερις ενδιάμεσες απεικονίσεις των προγραμμάτων, οι οποίες μπορούν να χρησιμοποιηθούν από διαφορετικές τεχνικές ανάλυσης. Κάθε μια από αυτές τις απεικονίσεις έχει διαφορετικό επίπεδο αφαίρεσης, το οποίο προσφέρει διαφορετικά οφέλη. Αυτές οι απεικονίσεις είναι:

- Baf
- Grimp
- Jimple
- Shimple

Το Soot μπορεί να εκτελεστεί από την γραμμή εντολών χρησιμοποιώντας από την γενική μορφή της εντολής

```
java [□π□□□□□□ java] soot.Main [□π□□□□□□ Soot] □□□□□ □□□□□□
```

Στις επιλογές της Java μπορεί να καθοριστούν διάφορες αναλύσεις και ρυθμίσεις του εργαλείου όπως

- ένα σύνολο επιλογών για τον έλεγχο και την διαχείριση της εισόδου του `Soot`
- η επιλογή των ζητούμενων αποτελεσμάτων και σε τι μορφή θα εμφανιστούν
- την ενδιάμεση απεικόνιση του προγράμματος που θα δημιουργήσει το εργαλείο
- ...

3 Λειτουργίες

Σε αυτή την ενότητα θα καθοριστούν οι λειτουργίες που θα υποστηρίζεί η πλατφόρμα TRACER αλλά και τα λειτουργικά της αρθρώματα συμφωνα με την ανάλυση των απαιτήσεων που έγινε στο παραδοτέο 1.2.

3.1 Λειτουργίες Πλατφόρμας TRACER για την διαχείριση τον αρθρωμάτων ασφαλείας.

Σύμφωνα με την λειτουργική απαίτηση 4.1 που καθορίστηκε κατά την ανάλυση των απαιτήσεων, η πλατφόρμα θα πρέπει να διαθέτει μια υποδομή ώστε να είναι επεκτάσιμη μεσω λειτουργικών αρθρωμάτων. Όπως είδαμε στην ενότητα 2, υπάρχει μια ποικιλία εργαλείων που μπορούν να ενσωματωθούν στην πλατφόρμα. Προκειμένου να επιτευχθεί η ενσωμάτωση διαφορετικών εργαλείων θα πρέπει η διαχείριση τους να είναι διαφανής (transparent) από την πλατφόρμα. Έτσι θα πρέπει να οριστεί ένα μια προγραμματιστική διεπαφη (**API!**) που θα επιτρέπει βασικές λειτουργίες διαχείρισής, όπως την ενεργοποίηση ενός plugin. Κάθε plugin οφείλει να υλοποιήσει αυτές τις λειτουργίες προκειμένου να μπορεί να ενσωματωθεί στην πλατφόρμα.

Μερικές από τις λειτουργίες για την διαχείριση κάθε αρθρώματος είναι η λειτουργία της *ενεργοποίησης* και της *απενεργοποίησης* ενός αρθρώματος. Κατά την ενεργοποίηση, κάθε plugin γίνεται διαθέσιμο πρός εκτέλεση από την πλατφόρμα. Κατά την απενεργοποίηση συμβαίνει το αντιθετό. Επίσης η πλατφόρμα θα πρέπει να είναι σε θέση να γνωρίζει το είδος της τρωτότητας που κάθε άρθρωμα θα εξετάζει. Άρα θα πρέπει το άρθρωμα να δίνει μια περιγραφή των ενεργειών που επιτελέι καθώς και των τρωτοτήτων που εξετάζει. Με αυτές τις λειτουργίες δημιουργείται μια υποδομή υποστήριξης αρθρωμάτων, όπως ορίζει η λειτουργική απαίτηση 4.1

Οι πιο ουσιώδεις λειτουργίες ενός αρθρώματος, είναι αυτές που αλληλεπιδρούν με τον ίδιο τον κώδικα του υπό εξέταση συστήματος. Κάθε άρθρωμα, εξετάζει ενα δοθέν αρχείο ή σύστημα για τυχόν τρωτότητες της κατηγορίας που εντοπίζει (πχ. buffer overflows).

Αν βρεθεί κάποια τρωτότητα θα πρέπει να την καταγράψει, και στο τέλος της εξέτασης να επιστρέψει το σύνολο των τρωτοτήτων που εντόπισε στην πλατφόρμα. Από εδώ προκύπτει η λειτουργία της *εκτέλεσης* της ανάλυσης από το plugin, καλύπτοντας ετσι την λειτουργική απαίτηση 3.1 για έλεγχο του κώδικα για αδυναμίες.Το άρθρωμα είναι επιφορτισμένο με το έλεγχο του κώδικα που του δόθηκε ως είσοδος και καλείται να απαντήσει στην πλατφόρμα αν ο κώδικας αυτός περιέχει ή όχι τις τρωτότητες που εξετάζει. Εσωτερικά, το άρθρωμα εκκινεί εξωτερικά εργαλεία στατικής ανάλυσης (αν το άρθρωμα λειτουργεί ως wrapper εξωτερικού εργαλείου) ή υλοποιεί δικές του μεθόδους (αν πρόκειται για custom plugin).

Επίσης προκύπτει η λειτουργία της *καταγραφής* όλων των προβλημάτων ασφαλείας που εντοπίστηκαν, καλύπτοντας έτσι την λειτουργική απαίτηση 3.2. Σε αυτή τη λειτουργία τα προβλήματα ασφαλείας που εντοπίστηκαν επιστρέφονται στην πλατφόρμα σύμφωνα με ένα προκαθορισμένο τρόπο. Συγκεκριμένα το άρθρωμα θα πρέπει να καταγράψει πληροφορίες όπως το είδος της τρωτότητας που εντοπίστηκε, το σημείο του κώδικα που εντοπίστηκε, το πόσο σημαντική είναι η συγκεκριμένη τρωτότητα για την ασφάλεια του συστήματος κ.α.

Πέρα από την καταγραφή των προβλημάτων, η πλατφόρμα θα είναι σε θέση να προτείνει λύσεις για τις τρωτότητες που εντόπισε. Συγκεκριμένα θα προτείνει βιβλιοθήκες που μπορούν να καταπολεμήσουν τις καταγεγραμμένες αδυναμίες.

3.2 Ενσωμάτωση εξωτερικών εργαλείων

Όπως είδαμε και στην ενότητα 2, υπάρχει μια πληθώρα εργαλείων στατικής ανάλυσης. Προκειμένου να ενσωματωθούν αυτά τα εργαλεία στην πλατφόρμα, όπως ορίζει η λειτουργική απαίτηση 4.2, θα πρέπει να κατασκευαστούν αρθρώματα τα οποία θα λειτουργούν ως wrappers των εργαλείων αυτών. Οι wrappers είναι αρθρώματα που υλοποιούν τις λειτουργίες της προγραμματιστικής διεπαφής που ορίζει η πλατφόρμα, καλώντας εσωτερικά το εκάστοτε εργαλείο και προωθώντας (ύστερα από επεξεργασία) τα αποτελέσματα του στην πλατφόρμα.

Προκειμένου να διευκολυνθεί ο προγραμματιστής στην κατασκευή του αρθρώματος που θα καλέι το εξωτερικό εργαλείο (wrapper), η πλατφόρμα προσφέρει ένα μηχανισμό κατασκευής ενός σκελετού του αρθρώματος (template). Το template αυτό θα περιέχει όλες τις μεθόδους που πρέπει να υλοποιηθούν προκειμένου το άρθρωμα να μπορέσει να ενσωματωθεί και να λειτουργεί σωστά στην πλατφόρμα. Πάνω σε αυτό το plugin, ο προγραμματιστής θα γράψει τον κατάλληλο κώδικα για να καλέσει το εξωτερικό εργαλείο και να επιστρέψει τα αποτελέσματα του στην πλατφόρμα με μορφή κατανοητή από αυτήν.

Η χρήση templates, απλοποιεί αρκετά την δημιουργία νέων αρθρωμάτων, καθώς ο προγραμματιστής θα είναι επιφορτισμένος μόνο με τη συγγραφή του κώδικα που θα καλεί το εξωτερικό εργαλείο, αποφεύγοντας έτσι πιθανά προβλήματα integration με την πλατφόρμα. Το template ενός αρθρώματος εξασφαλίζει πως η επικοινωνία μεταξύ του αρθρώματος και της πλατφόρμας γίνεται ορθά, σύμφωνα με τον προκαθορισμένο τρόπο. Η κατασκευή templates θα χρησιμοποιηθεί και στην περίπτωση που ο προγραμματιστής θα θελήσει να ενσωματώσει custom plugins, προσαρμοσμένα στην φύση της εφαρμογής που εξετάζεται.

4 Ορισμός API

4.1 Εισαγωγή

Στην ενότητα αυτή θα περιγραφεί ο σχεδιασμός των προγραμματιστικών διεπαφών για την αναπαράσταση των οντοτήτων που διαχειρίζεται η πλατφόρμα TRACER, την επικοινωνία της πλατφόρμας με εφαρμογές πελάτη (client-side applications) και την δημιουργία αρθρωμάτων (plugins) για την ενσωμάτωση εξωτερικών εργαλείων ανάλυσης τρωτοτήτων.

Για την ανάπτυξη της πλατφόρμας επιλέχθηκε να χρησιμοποιηθεί το Alitheia Core, ο πυρήνας της πλατφόρμας λογισμικού Alitheia. Το Alitheia είναι μια ολοκληρωμένη πλατφόρμα για την υποστήριξη της έρευνας στον τομέα της τεχνολογίας λογισμικού, η οποία είχε αναπτυχθεί στο πλαίσιο του έργου SQOOSS.

Το Alitheia Core παρέχει ένα μοντέλο ασφάλειας για την πρόσβαση στις υπηρεσίες και τους πόρους της πλατφόρμας μέσω ενός **URL!** (**URL!**) που αντιστοιχεί στον καθένα. Η χρήση **URL!**s για την αναπαράσταση των πόρων και των υπηρεσιών της πλατφόρμας επιτρέπει παράλληλα τον έλεγχο πρόσβασης σε αυτές με εύκολο τρόπο. Κάθε **URL!** είναι συσχετισμένο με μια ή περισσότερες ομάδες χρηστών, και καθεμιά από αυτές με ένα σύνολο αδειών και δικαιωμάτων. Έτσι μπορεί να υλοποιηθούν λεπτομερή σενάρια πρόσβασης στους πόρους για τους οποίους απαιτείται εξουσιοδότηση. Ένας χρήστης

ανήκει σε μια ή περισσότερες ομάδες χρηστών, επιτρέποντας ευκολότερη διαχείριση των δικαιωμάτων, αφού αυτά ανατίθενται ανά ομάδα. Το μοντέλο ασφάλειας παρέχει επίσης τη δυνατότητα αποθήκευσης χρηστών και ομάδων χρηστών, αλλά δεν παρέχει κάποιο επίπεδο διαχείρισης των συνόδων του καθενός (session management), αφού αυτό εξαρτάται από την εκάστοτε υπηρεσία ή εφαρμογή που χρησιμοποιεί το μοντέλο ασφάλειας. Το ίδιο ισχύει και για επιπλέον λειτουργικότητα που μπορεί να απαιτείται κατά περίπτωση, όπως την εφαρμογή πολιτικών για τα συνθηματικά των χρηστών, κλπ.

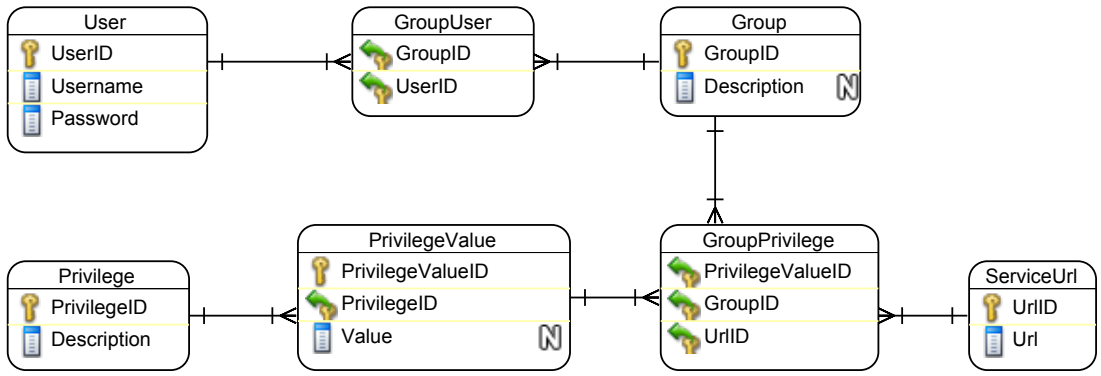
Στο μοντέλο ασφάλειας του Alitheia Core ένα δικαίωμα είναι συνώνυμο με μια ενέργεια που εκτελείται στην πλατφόρμα. Ένα δικαίωμα για παράδειγμα μπορεί να είναι η δυνατότητα να ανακτήσει κάποιος χρήστης τη λίστα των χρηστών της πλατφόρμας, ή να ενημερώσει τα στοιχεία ενός προφίλ ασφάλειας. Κάθε δικαίωμα μπορεί να έχει διακριτές τιμές, οι οποίες επιτρέπουν ακόμα πιο λεπτομερή έλεγχο πρόσβασης. Για παράδειγμα, έστω το παρακάτω **URL!**

```
svc://tracer.securityprofile.management
```

Συνδυάζοντάς το με ένα privilege με όνομα action και πιθανές τιμές ``view'', ``create'' και ``delete'' μπορούν να οριστούν πολύ λεπτομερείς έλεγχοι για την πρόσβαση στην παραπάνω υπηρεσία, όπως φαίνεται στο επόμενο παράδειγμα:

```
svc://tracer.securityprofile.management?action=create
```

Το schema της βάσης δεδομένων στο οποίο στηρίζεται το μοντέλο ασφάλειας του Alitheia παρουσιάζεται στο Σχήμα 1.



Σχήμα 1: Database Schema του μοντέλου ασφάλειας του Alitheia

4.2 Προγραμματιστική διεπαφή οντοτήτων πλατφόρμας

Οι λειτουργίες που θα πρέπει να υποστηρίζονται από την πλατφόρμα TRACER όπως περιγράφονται στην ενότητα 3 επιβάλλουν τη δημιουργία ενός συνόλου κλάσεων οι οποίες θα χρησιμοποιούνται για την αναπαράσταση των αντίστοιχων οντοτήτων. Οι οντότητες αυτές αφορούν:

- την αναπαράσταση των προβλημάτων ασφάλειας τις οποίες θα εντοπίζει η πλατφόρμα

- τη συσχέτιση βιβλιοθηκών ασφάλειας με τα προβλήματα για τα οποία παρέχουν θωράκιση
- τον ορισμό και τη διαχείριση προφίλ ασφάλειας και λιστών παρακολούθησης αρχείων

Δεδομένου ότι το Alitheia Core στο οποίο θα βασιστεί η υλοποίηση της πλατφόρμας έχει υλοποιηθεί στη γλώσσα προγραμματισμού Java, οι διεπαφές αυτές θα ενσωματωθούν σε μια βιβλιοθήκη κώδικα που θα μπορεί να χρησιμοποιείται από τις διάφορες εφαρμογές που θα παρέχουν τη διεπαφή χρήστη με την πλατφόρμα. Η διεπαφή αυτή θα χρησιμοποιείται επίσης από την προγραμματιστική διεπαφή επικοινωνίας της πλατφόρμας με τις εφαρμογές πελάτη. Η διεπαφή αυτή περιγράφεται ακολούθως με τη βοήθεια διαγραμμάτων κλάσεων, όπως έχει προκύψει από το παραδοτέο 2.1.

4.3 Προγραμματιστική διεπαφή επικοινωνίας της πλατφόρμας με εφαρμογές πελάτη

Για την επικοινωνία της πλατφόρμας TRACER με τον εξωτερικό κόσμο και τις εφαρμογές πελάτη η πλατφόρμα θα παρέχει ένα κεντρικοποιημένο σημείο πρόσβασης στη λειτουργικότητα που υλοποιεί μέσω μιας διεπαφής υπηρεσιών web (web service interface). Το web service θα παρέχει όλες τις μεθόδους που είναι απαραίτητες για την πραγματοποίηση των λειτουργιών που περιγράφονται στην ενότητα 3 και θα συνοδεύεται από ένα έγγραφο WSDL που θα περιγράφει τα δεδομένα που ανταλλάσονται σε κάθε κλήση μεθόδου. Η χαλαρή σύζευξη μεταξύ της πλατφόρμας και των εφαρμογών πελάτη θα επιτευχθεί με τη χρήση απομακρυσμένων κλήσεων των μεθόδων που παρέχει το web service (remote procedure calls). Η χρήση μιας τέτοιας τεχνολογίας που βασίζεται σε ανοιχτά πρότυπα θα διευκολύνει την ανάπτυξη των διαφόρων διεπαφών για την πλατφόρμα, αλλά και την ανάπτυξη εφαρμογών από τρίτους κατασκευαστές που θα μπορούν να δημιουργήσουν δικές τους λύσεις βασισμένες στην πλατφόρμα.

Ο σχεδιασμός της διεπαφής επικοινωνίας βασίζεται στην αναπαράσταση υπηρεσιών, πόρων (resources) και λειτουργιών / μεθόδω της πλατφόρμας με τη μορφή **URL!**s, προκειμένου αυτές να καταστούν προσβάσιμες μέσω μιας διεπαφής που ακολουθεί το πρότυπο REST. Η προσέγγιση REST επιτρέπει την ενοποιημένη αναπαράσταση πόρων σε ένα υπολογιστικό σύστημα. Προκειμένου να χρησιμοποιήσουν τους πόρους αυτούς, τα δομικά στοιχεία (components) του συστήματος επικοινωνούν μέσω ενός διαδεδομένου πρωτοκόλλου (πχ HTTP) ανταλλάσσοντας μεταξύ τους έγγραφα που αναπαριστούν τους υπολογιστικούς πόρους. Η επιλογή αυτής της προσέγγισης βασίστηκε και στο γεγονός ότι το μοντέλο ασφάλειας του Alitheia στηρίζεται στην ίδια τεχνική.

Η ευελιξία που προσφέρει η χρήση **URL!**s για την περιγραφή των διαφόρων υπηρεσιών της πλατφόρμας παρέχει τη δυνατότητα να περιγραφούν με απλό τρόπο όλες οι λειτουργίες (μέθοδοι) που αυτές παρέχουν και οι παράμετροι που απαιτούνται για να γίνει αυτό. Για παράδειγμα, αν η υπηρεσία διαχείρισης των χρηστών του συστήματος αναπαρασταθεί με το **URL!**

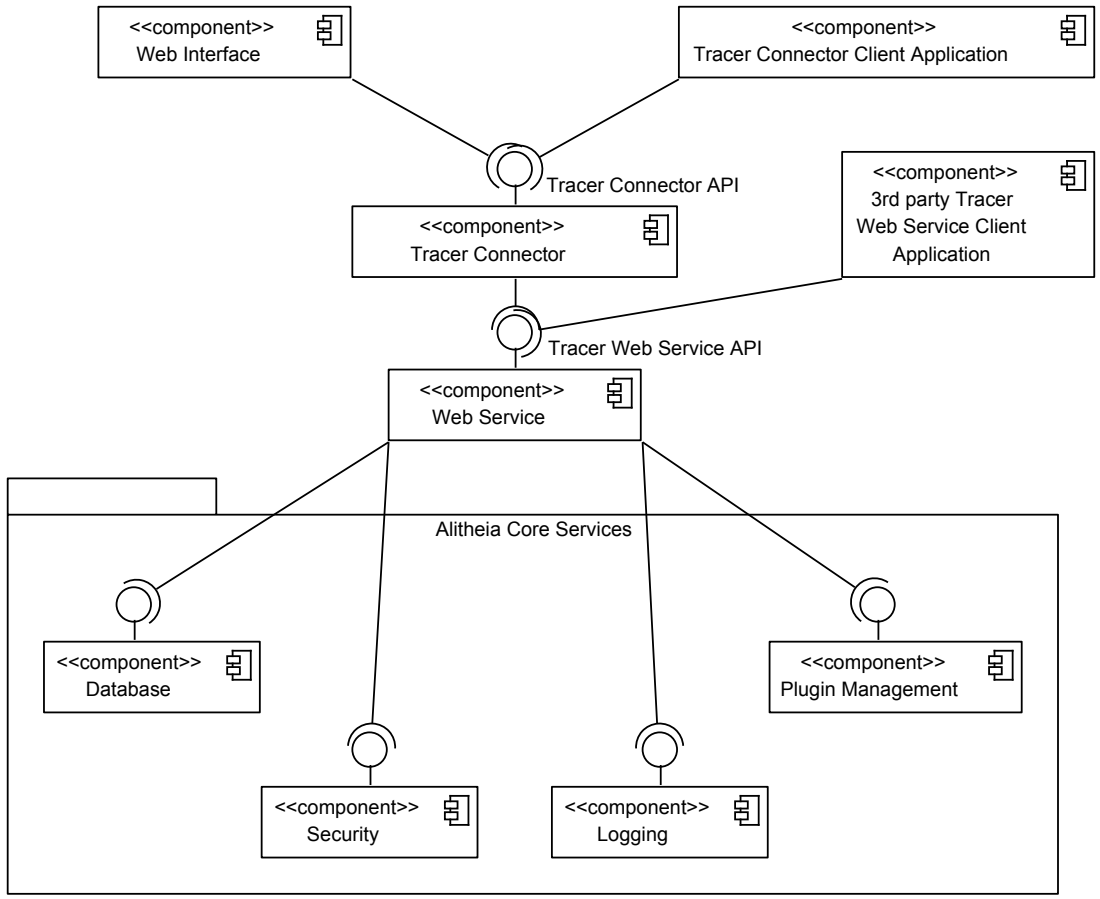
```
svc://tracer.usermanagement
```

τότε η λειτουργία της προσθήκης ενός νέου χρήστη μπορεί να παρασταθεί με την ακόλουθη μορφή:

```
svc://tracer.usermanagement?act=adduser
```

Η παραπάνω τεχνική δίνει τη δυνατότητα να κρυφτούν πολλές από τις λεπτομέρειες του web service **API!** που θα παρέχει η πλατφόρμα πίσω από κάποια ενδιάμεση βιβλιοθήκη. Ο σκοπός ύπαρξης της βιβλιοθήκης αυτής είναι να υλοποιήσει κοινή λειτουργικότητα όπως η σύνδεση με το web service, η διαχείριση των συνόδων των χρηστών (session management), και ο αυτοματοποιημένος έλεγχος των μηνυμάτων που ανταλλάσσονται μεταξύ του web service και της εφαρμογής-πελάτη. Με άλλα λόγια, η ενδιάμεση αυτή βιβλιοθήκη θα παρέχει ένα απλουστευμένο **API!** για τα web services που θα παρέχει η πλατφόρμα, το οποίο θα μεταφράζει τα αποτελέσματα των κλήσεων σε αντικείμενα τα οποία μπορούν να χρησιμοποιηθούν απευθείας από την εφαρμογή-πελάτη.

Η αρχιτεκτονική και οι αλληλεπιδράσεις ανάμεσα στις διεπαφές για την επικοινωνία με τις εφαρμογές πελάτη παρουσιάζεται στο Σχήμα 2. Στο σχήμα αυτό φαίνεται ξεκάθαρα πως μια εφαρμογή πελάτη μπορεί να χρησιμοποιήσει είτε απευθείας το web service που θα παρέχει η πλατφόρμα TRACER είτε να χρησιμοποιήσει την ενδιάμεση βιβλιοθήκη.



Σχήμα 2: Component diagram των προγραμματιστικών διεπαφών TRACER - εφαρμογών πελάτη

4.4 Προγραμματιστική διεπαφή αρθρωμάτων ενσωμάτωσης εξωτερικών εργαλείων

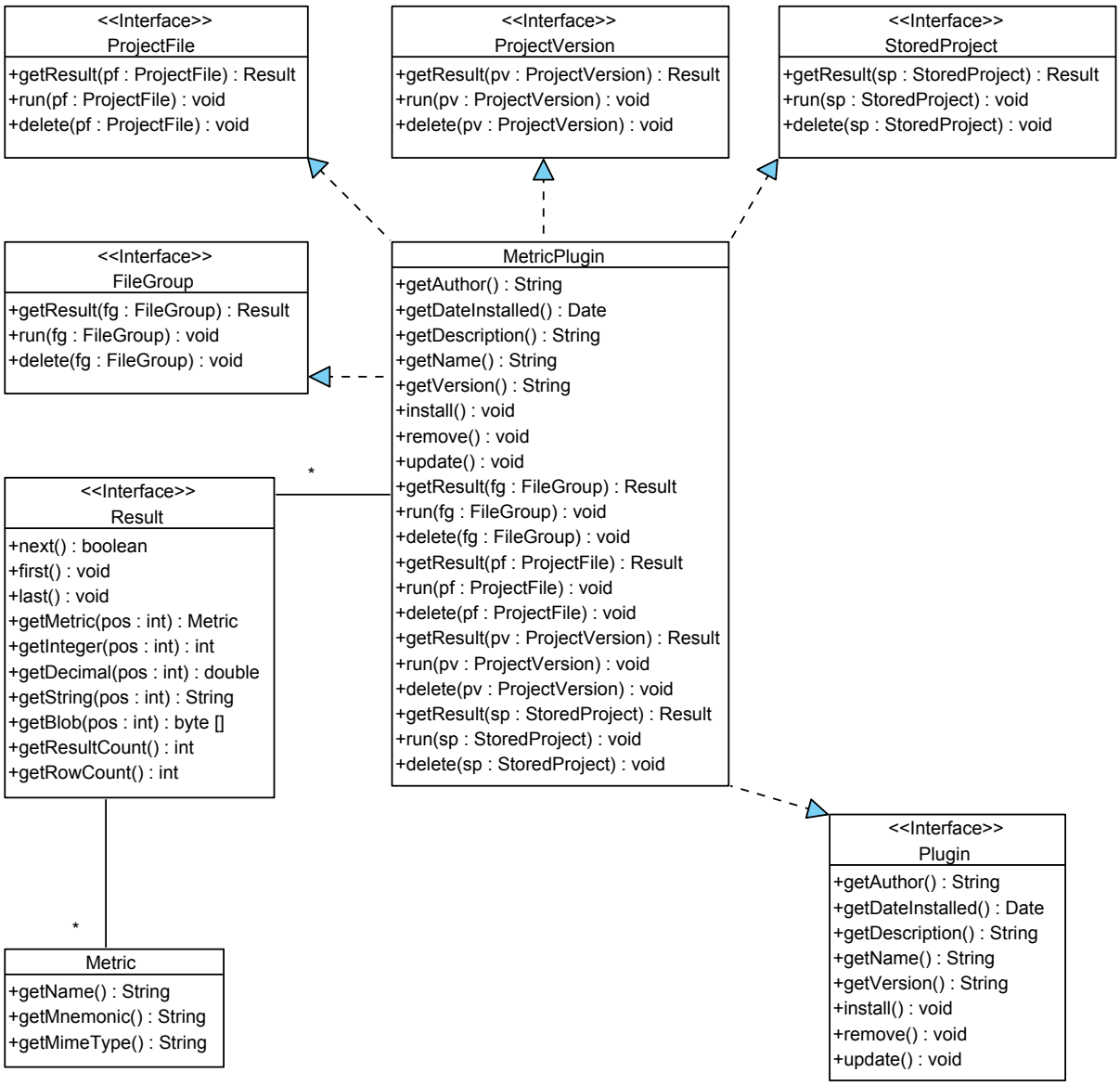
Η πλειοψηφία των εργαλείων που είναι διαθέσιμα για τον εντοπισμό τρωτοτήτων στο λογισμικό, όπως προκύπτει από την ενότητα 2, διαφέρει σημαντικά ως προς τον τρόπο λειτουργίας αλλά έχει ένα κοινό χαρακτηριστικό. Τα εργαλεία αυτά μπορούν να εκτελεστούν σε batch mode, χωρίς δηλαδή να χρειάζεται η αλληλεπίδραση μαζί τους από κάποιο γραφικό περιβάλλον. Τυπικά ένα τέτοιο εργαλείο δέχεται διάφορες παραμέτρους που καθορίζουν την ανάλυση που θα πραγματοποιηθεί και τα αρχεία (πηγαίου κώδικα ή εκτελέσιμα) στα οποία αυτή θα εστιάσει, και παράγει αποτελέσματα με τη μορφή κειμένου. Τα αποτελέσματα αυτά είτε τυπώνονται στο προκαθορισμένο ρεύμα εξόδου (standard output stream) ή αποθηκεύονται σε κάποιο αρχείο σε human ή machine readable format.

Τα παραπάνω χαρακτηριστικά κάνουν σχετικά εύκολη τη δημιουργία κάποιου αρθρώματος λογισμικού για την πλατφόρμα TRACER που να τρέχει ένα από αυτά τα εργαλεία, χρησιμοποιώντας ως πρότυπο το αρχέτυπο του αρθρώματος λογισμικού που παρέχεται από την πλατφόρμα για τη δημιουργία plugins που υπολογίζουν μετρικές. Ωστόσο, επειδή αναπόφευκτα κάποια κομμάτια της λειτουργικότητας αυτών των plugins θα είναι επικαλυπτόμενη, αποφασίστηκε η δημιουργία ενός νέου αρχέτυπου που θα προσφέρει αυτή την επιπλέον κοινή λειτουργικότητα μέσω ενός **API!** ώστε να μη χρειάζεται να υλοποιείται από την αρχή για κάθε plugin.

Η λειτουργία των plugins που χρησιμοποιούν εξωτερικά εργαλεία για την εύρεση τρωτοτήτων πρακτικά αφορά τις παρακάτω επιμέρους εργασίες:

- Τον καθορισμό γενικών παραμέτρων για την εκτέλεση του εργαλείου, όπως πχ της τοποθεσίας στην οποία αυτό είναι αποθηκευμένο και των ρυθμίσεων του περιβάλλοντος εκτέλεσης,
- Τη δημιουργία της κατάλληλης εντολής για την κλήση του εξωτερικού εργαλείου, ανάλογα με το είδος της ανάλυσης που πρέπει να πραγματοποιηθεί και τα αρχεία στα οποία θα πρέπει να εκτελεστεί,
- Την εκτέλεση του εξωτερικού εργαλείου και τον έλεγχο του κατά πόσον αυτή τερματίστηκε χωρίς σφάλματα,
- Την ανάκτηση των αποτελεσμάτων από την εκτέλεση του εργαλείου, είτε αυτή γίνεται από το τυπικό ρεύμα εξόδου της εφαρμογής είτε από κάποιο αρχείο το οποίο αυτή δημιουργεί, και τέλος
- Την ανάλυση των αποτελεσμάτων του εργαλείου και την καταχώρηση των τρωτοτήτων που εντοπίστηκαν στο σύστημα.

Τα plugins στην πλατφόρμα TRACER βασίζονται στην υποδομή που παρέχει το Alitheia Core για την δημιουργία plugins τα οποία υπολογίζουν μετρικές ποιότητας λογισμικού και αποθηκεύουν τα αποτελέσματά τους στη βάση δεδομένων της πλατφόρμας. Όπως φαίνεται στο Σχήμα 3, εννοιολογικά κάθε plugin είναι μια κλάση που υλοποιεί ορισμένες διεπαφές οι οποίες καθορίζουν την αλληλεπίδραση που θα έχει με την υπόλοιπη



Σχήμα 3: Class diagram για την υποστήριξη plugins στο Alitheia Core

πλατφόρμα. Υλοποιώντας την αντίστοιχη διεπαφή ένα plugin μπορεί να εκτελεστεί αυτόματα για την επεξεργασία του συνόλου των δεδομένων που υποστηρίζει. Έτσι, υλοποιώντας τις διεπαφές `StoredProject`, `ProjectVersion`, `ProjectFile`, `FileGroup` ένα plugin μπορεί να εκτελέσει την επεξεργασία του αντίστοιχου συνόλου δεδομένων. Επομένως κάθε plugin χρειάζεται να υλοποιήσει μόνο τα κατάλληλα interfaces ανάλογα με το είδος των δεδομένων που μπορεί να χειριστεί. Επομένως ένα plugin που μπορεί να επεξεργαστεί μόνο μεμονωμένα αρχεία θα πρέπει να υλοποιεί μόνο τη διεπαφή `ProjectFile`.

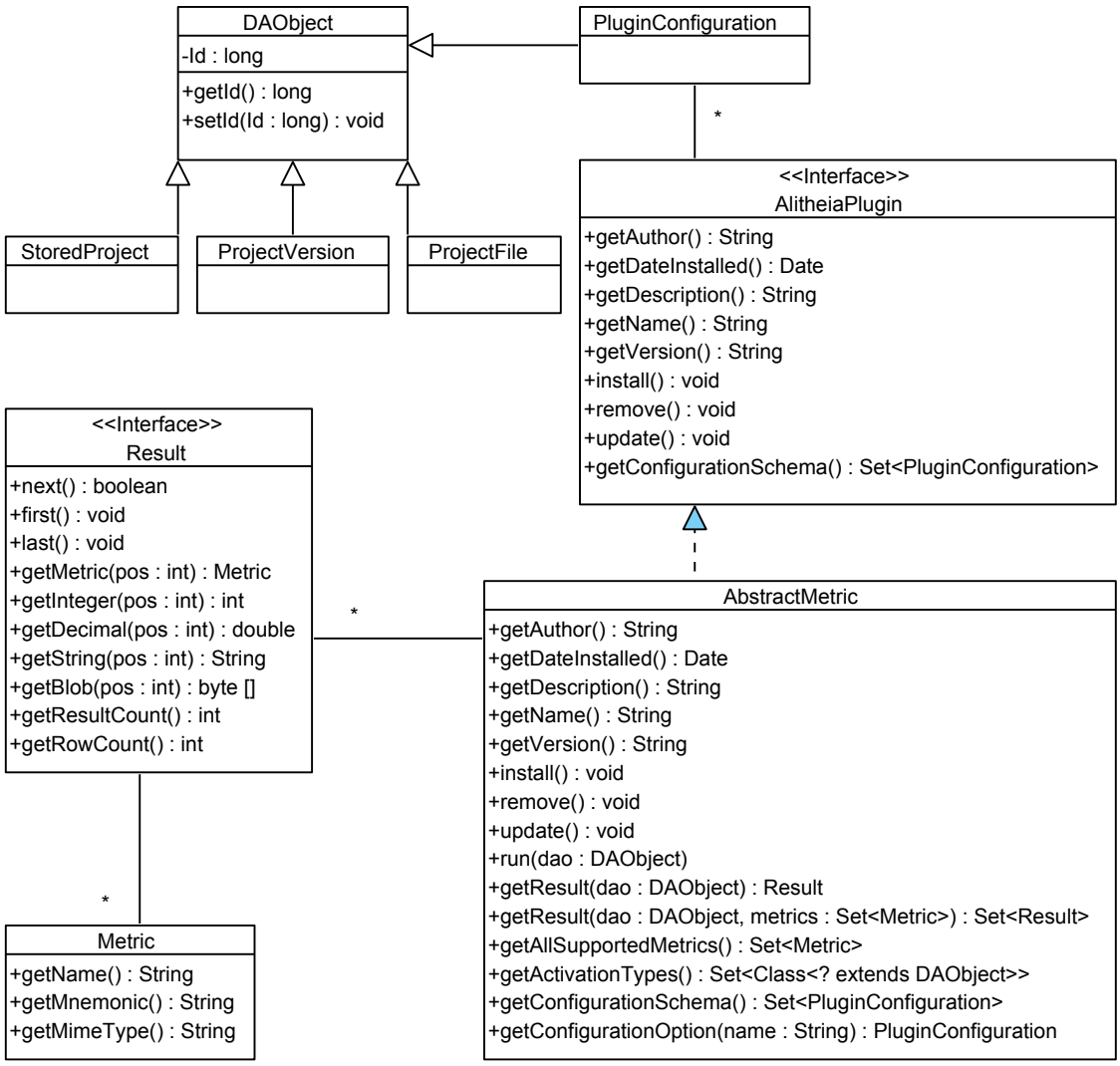
Η διεπαφή `Plugin` περιλαμβάνει τη λειτουργικότητα που πρέπει να διαθέτει κάθε plugin και αφορά στην διαχείριση και ανάκτηση πληροφοριών σχετικά με αυτό, την εγκατάσταση, απεγκατάσταση και ενημέρωσή του. Χρησιμοποιείται κυρίως εσωτερικά από την πλατφόρμα. Επειδή κάθε plugin είναι υπεύθυνο για την διαχείριση των δεδομένων του, αν χρειάζεται η δημιουργία νέων πινάκων στη βάση δεδομένων και η ενημέρωσή

τους κατά τη διαδικασία ενημέρωσης της έκδοσης του plugin, θα πρέπει η λειτουργικότητα αυτή να υλοποιηθεί στις αντίστοιχες μεθόδους της διεπαφής `Plugin`. Παράλληλα, το plugin θα πρέπει να φροντίζει για τη διατήρηση της σχεσιακής ακεραιότητας των δεδομένων του (referential integrity), και να κάνει διαθέσιμα στην υπόλοιπη πλατφόρμα αντικείμενα (Data Access Object classes) για την αποθήκευση δεδομένων στους πίνακες που διαχειρίζεται το ίδιο.

Τα αποτελέσματα που προκύπτουν από την επεξεργασία των δεδομένων που αποθηκεύονται στην πλατφόρμα γίνονται διαθέσιμα μέσω της διεπαφής `Result`, η οποία ακολουθεί εν μέρει το σχεδιασμό της κλάσης `ResultSet` που παρέχεται από το **JDBC!** (**JDBC!**). Ένα `Result` περιγράφει μια δομή που κρατά ένα τοπικό αντίγραφο ενός συνόλου δεδομένων που προκύπτουν από την επεξεργασία ενός plugin. Περιέχει ένα σύνολο από γραμμές (rows) κάθε μια απο τις οποίες περιέχει ένα σύνολο στηλών, η κάθε μια απο τις οποίες περιέχει την τιμή μιας μετρικής για ένα συγκεκριμένο τεχνούργημα (artifact) ενός έργου λογισμικού (πχ για ένα `ProjectVersion`, ένα `ProjectFile`, ή ένα `FileGroup`). Για την ανάκτηση της πραγματικής τιμής χρησιμοποιούνται οι μέθοδοι `getInteger`, `getString`, `getDecimal`, `getBlob`, που δέχονται ως όρισμα τον αριθμό της στήλης και επιστρέφουν την τιμή μεταφρασμένη αντίστοιχα στον κατάλληλο τύπο δεδομένων. Η μέθοδος `getMetric` επιστρέφει μια αναφορά σε ένα αντικείμενο τύπου `Metric` που περιγράφει την συγκεκριμένη μετρική που περιέχεται στην επιλεγμένη στήλη, ενώ οι μέθοδοι `next`, `first` και `last` επιτρέπουν την πλοήγηση ανάμεσα στις γραμμές των αποτελεσμάτων.

Σε επίπεδο υλοποίησης και προκειμένου να απλοποιηθεί η διεπαφή των plugins οι διεπαφές `StoredProject`, `ProjectVersion`, `ProjectFile`, και `FileGroup` αντικαταστάθηκαν από αντίστοιχες κλάσεις με κοινή ρίζα την αφηρημένη κλάση `DAObject` η οποία χρησιμοποιείται για να περιγράψει ένα οποιοδήποτε τεχνούργημα (artifact) που απαρτίζει το λογισμικό, και δεν περιορίζεται στις τέσσερις παραπάνω κατηγορίες. Αντίστοιχα η διεπαφή των Plugins δέχεται ως ορίσματα αντικείμενα τύπου `DAObject`, οπότε ο δημιουργός του plugin μπορεί να ορίσει περισσότερες από μια μεθόδους με το ίδιο όνομα για να καθορίσει συγκεκριμένη συμπεριφορά ανάλογα με τον τύπο του τεχνουργήματος που πρέπει να επεξεργαστεί το plugin. Η κλάση `AbstractMetric` αποτελεί τη βασική αφηρημένη κλάση για τη δημιουργία ενός plugin. Η κλάση `PluginConfiguration` επιτρέπει σε ένα plugin να αποθηκεύει παραμέτρους για την εκτέλεσή του, οι οποίες μπορούν να ανακτηθούν με τη χρήση των μεθόδων `getConfigurationSchema` και `getConfigurationOption`. Το διάγραμμα κλάσεων της υλοποίησης για την υποστήριξη των plugins στο Alitheia Core παρουσιάζεται στο Σχήμα 4, στο οποίο για λόγους απλότητας παραλείπονται τα attributes και οι μέθοδοι των κλάσεων που κληρονομούν από την `DAObject`.

Το Alitheia Core παρέχει ένα αρχέτυπο (archetype) για τη δημιουργία plugins που υλοποιεί τις λειτουργίες που περιγράφτηκαν πιο πάνω, και χρησιμεύει ως σκελετός για την υλοποίηση νέων plugins. Στο πλαίσιο του TRACER θα δημιουργηθεί ένα νέο αρχέτυπο plugin το οποίο θα είναι προσανατολισμένο στην εύκολη καταχώρηση τρωτοτήτων που έχουν εντοπιστεί στο λογισμικό από εξωτερικά εργαλεία, όπως περιγράφτηκε παραπάνω, χρησιμοποιώντας τις διεπαφές για την αναπράσταση τρωτοτήτων που περιγράφονται στην ενότητα 4.2. Πρακτικά θα υλοποιηθούν ως πρότυπες μέθοδοι (template methods) οι επιμέρους εργασίες που πρέπει να εκτελέσει ένα τέτοιο plugin όταν καλείται η αντίστοιχη μέθοδος `run` του plugin. Αυτό σημαίνει πως ο δημιουργός ενός τέτοιου

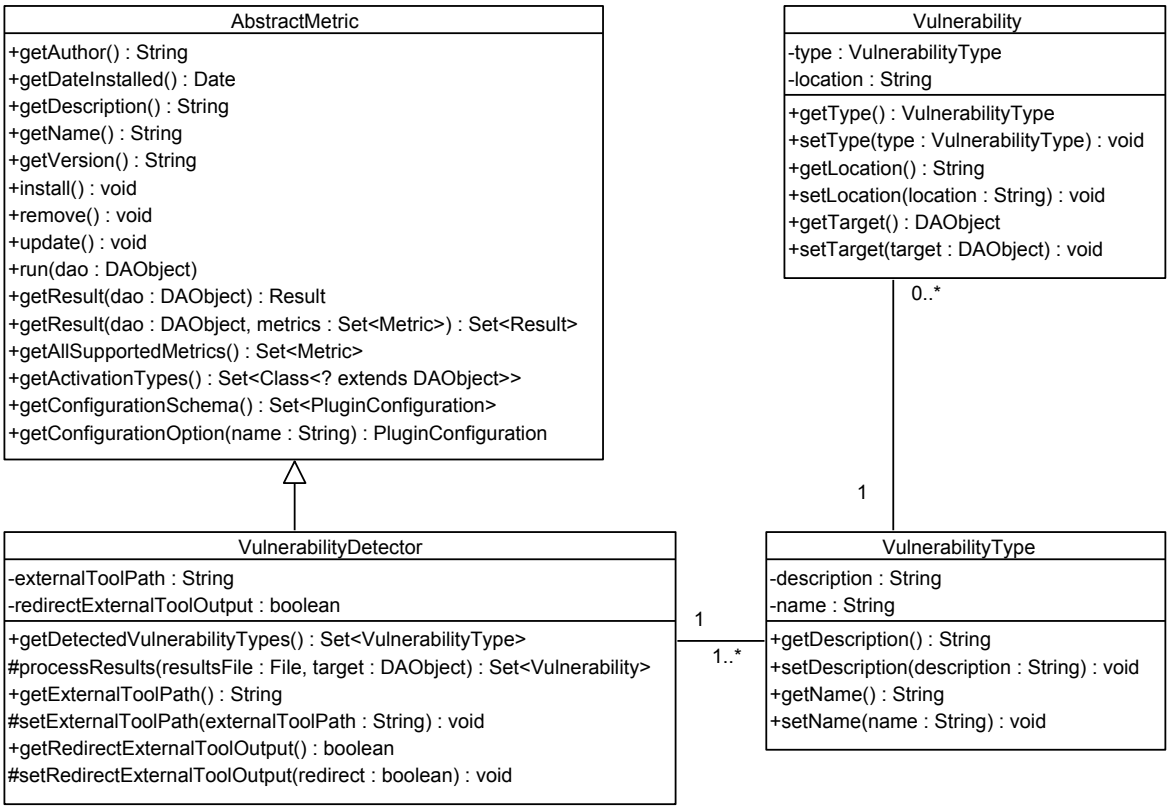


Σχήμα 4: Class diagram της υλοποίησης της υποδομής plugins στο Alitheia Core

plugin θα έχει τη δυνατότητα αν χρειάζεται να εκτελέσει κάποια πιο περίπλοκη διαδικασία που περιλαμβάνει περισσότερα στάδια (πχ μεταγλώττιση, pre-processing, κλπ.) για τον εντοπισμό τρωτοτήτων, να έχει τη δυνατότητα να την εκτελέσει υλοποιώντας την αντίστοιχη μέθοδο `run`. Παράλληλα, στην πλειοψηφία των περιπτώσεων, θα χρειαστεί κανείς να υλοποιήσει μόνο τις μεθόδους που αφορούν στην εκτέλεση του εξωτερικού εργαλείου με τις κατάλληλες παραμέτρους και την ανάλυση των αποτελεσμάτων του.

Η αρχιτεκτονική που επιτρέπει τα παραπάνω παρουσιάζεται στο διάγραμμα κλάσεων που βρίσκεται στο Σχήμα 5. Πιο αναλυτικά, η βασική κλάση που υλοποιεί τις τυπικές λειτουργίες ενός Vulnerability Detector plugin το οποίο εκτελεί ένα εξωτερικό εργαλείο ανάλυσης τρωτοτήτων είναι η αφηρημένη κλάση `VulnerabilityDetector`. Αυτή είναι παιδί της κλάσης `AbstractMetric` που περιγράφτηκε προηγουμένως, και περιέχει υλοποιημένες ως template methods τις παραλλαγές της μεθόδου `run` επιτρέποντας στις συμπαγείς (concrete) κλάσεις που την κληρονομούν να εκτελέσουν τα διαφορετικά στάδια της ανίχνευσης τρωτοτήτων ανάλογα με τις ιδιαιτερότητες του εξωτερικού εργαλείου που χρησιμοποιεί η καθεμία αντίστοιχα. Η μέδοδος `getDetectedVulnerabilityTypes`

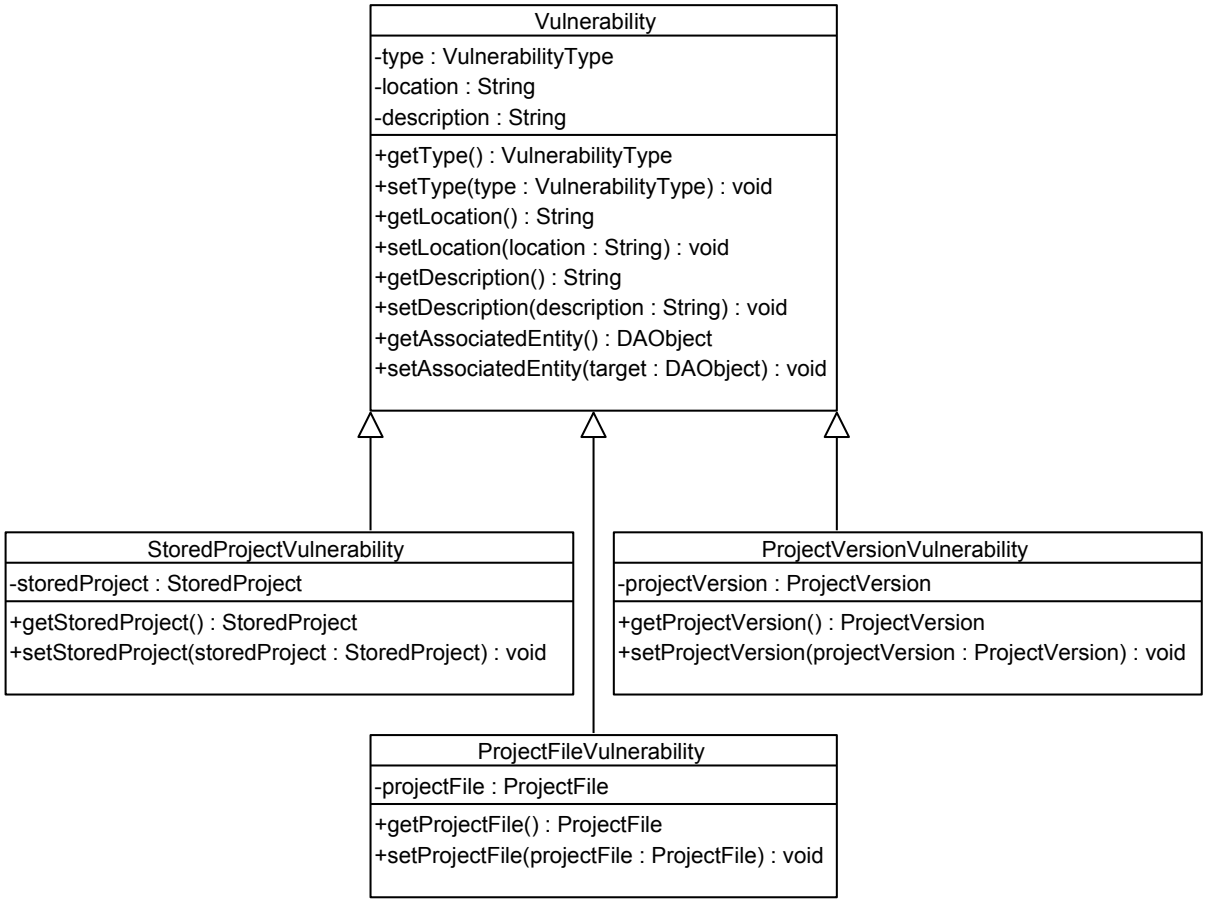
επιστρέφει το σύνολο των τύπων των τρωτοτήτων τις οποίες μπορεί να εντοπίσει το συγκεκριμένο plugin. Η μέθοδος `processResults` δέχεται ως όρισμα το αρχείο που έχει προκύψει από το εξωτερικό εργαλείο κατά την ανάλυση τρωτοτήτων (είτε αυτό δημιουργείται από το ίδιο το εργαλείο είτε μέσω ανακατεύθυνση του ρεύματος εξόδου του) και το αναλύει επιστρέφοντας το σύνολο των στιγμιοτύπων των τρωτοτήτων που εντοπίστηκαν. Αυτή κατά κύριο λόγο είναι η μέθοδος η οποία χρειάζεται να υλοποιηθεί από το εκάστοτε plugin ανάλογα με το εξωτερικό εργαλείο και το είδος της ανάλυσης που πραγματοποιείται.



Σχήμα 5: Class diagram της υποστήριξης για Vulnerability Detector plug-ins στην πλατφόρμα TRACER

Η αφηρημένη κλάση `Vulnerability` αναπαριστά τα συγκεκριμένα στιγμιότυπα μιας τρωτότητας που έχει εντοπιστεί από κάποιο `VulnerabilityDetector`. Προκειμένου να διευκολυνθεί η διαχείρισή τους, ανάλογα με τον τύπο του αντικειμένου στο οποίο πραγματοποιείται ο έλεγχος, συγκεκριμενοποιείται από τις κλάσεις `StoredProjectVulnerability`, `ProjectFileVulnerability` και `ProjectVersionVulnerability`, όπως παρουσιάζεται στο Σχήμα 6.

Ο πίνακας 1 συσχετίζει τις λειτουργικές απαιτήσεις που τέθηκαν στο παραδοτέο 1.2, στις αντίστοιχες λειτουργίες που θα πρέπει να προσφέρει η πλατφόρμα όπως περιγράφηκαν στην ενότητα 3 με τις τελικές κλήσεις των λειτουργιών αυτών μέσω **URL!**s. Οι δύο πρώτες στήλες απεικονίζουν τον αριθμό και την περιγραφή των λειτουργικών απαιτήσεων από το παραδοτέο 1.2. Στη συνέχεια, η τρίτη στήλη αντιστοιχίζει κάθε απαίτηση με την αντίστοιχη λειτουργία ή λειτουργίες που την καλύπτεί. Τέλος, η τέταρτη στήλη, μας δείχνει τον τρόπο κλήσης της εκάστοτε λειτουργίας, μέσω **URL!**. Αναλυτικότερα



Σχήμα 6: Class diagram της ιεραρχίας των τύπων τρωτοτήτων στην πλατφόρμα TRACER

5 Τεχνολογίες υλοποίησης

Αυτή η ενότητα περιγράφει τις τεχνολογίες ανοικτού κώδικα και τα καθιερωμένα πρότυπα που επιλέχθηκαν για την ανάπτυξη της πλατφόρμας TRACER. Η χρήση των τεχνολογιών αυτών επιτρέπει τη δημιουργία μιας στιβαρής πλατφόρμας για την ανάλυση πηγαίου κώδικα λογισμικού για κενά ασφάλειας και παρουσίαση των αποτελεσμάτων.

Το framework πάνω στο οποίο θα αναπτυχθεί η πλατφόρμα TRACER δεν είναι άλλο από το **OSGi!** (**OSGi!**), ενώ για την υποστήριξη της δημιουργίας web services θα χρησιμοποιηθεί το Apache EXtensible Interaction System 2 (Axis2). Περισσότερες λεπτομέρειες για το καθένα υπάρχουν στις ενότητες 5.1 και 5.2 αντίστοιχα, ενώ ο web container που θα χρησιμοποιηθεί για την υποστήριξη της web-based διεπαφής χρήστη είναι ο *Apache Tomcat* που περιγράφεται στην ενότητα 5.3.

5.1 Εισαγωγή στα OSGi και Equinox

Το **OSGi!** (**OSGi!**) Alliance[1] είναι ένας ανεξάρτητος οργανισμός που ορίζει και εξελίσσει ένα σύνολο προτύπων και προδιαγραφών για την ανάπτυξη και τη λειτουργία διαλει-

[1] `http://www.osgi.org`

τουργικών υπηρεσιών λογισμικού σε δικτυακά περιβάλλοντα. Ο πυρήνας των προδιαγραφών αυτών είναι το **OSGi!** Framework. Το **OSGi!** Framework χωρίζεται σε 4 διακριτά επίπεδα και παρέχει ένα προτυποποιημένο περιβάλλον για την εκτέλεση εφαρμογών.

Το πρότυπο **OSGi!** έχει υλοποιηθεί από ένα σημαντικό αριθμό παρόχων λογισμικού. Στο πλαίσιο του έργου χρησιμοποιείται μια υλοποίηση ανοιχτού κώδικα που καλύπτεται από την άδεια Eclipse Public License[2] (EPL), και ονομάζεται Equinox[3].

5.1.1 Το OSGi Framework

Το **OSGi!** Framework υλοποιεί ένα μοντέλο για τη δημιουργία δομικών στοιχείων λογισμικού (component model), με σκοπό την παροχή ενός πλαισίου λογισμικού που είναι γραμμένο σε καθαρή Java και επιτρέπει τη δυναμική ανάπτυξη (deployment) λειτουργικών μονάδων εφαρμογών λογισμικού. Αυτές οι μονάδες εφαρμογών ονομάζονται bundles και μπορούν να εγκατασταθούν, να ξεκινήσουν ή να σταματήσουν να λειτουγούν, να αναβαθμιστούν και να απεγκατασταθούν είτε τοπικά είτε απομακρυσμένα, χωρίς να χρειάζεται να ξεκινήσει ξανά η εφαρμογή που τις χρησιμοποιεί. Σε πρακτικό επίπεδο, ένα bundle είναι ένα αρχείο **JAR!** (**JAR!**) το οποίο εκτός από μεταγλωττισμένο κώδικα μπορεί να περιέχει επίσης άλλους πόρους όπως εικόνες, αρχεία XML, HTML, άλλα αρχεία **JAR!** κλπ. Περιέχει επίσης ένα manifest file που περιέχει πληροφορίες για τα περιεχόμενά του.

Τα τέσσερα επίπεδα στα οποία βασίζεται η δομή του **OSGi!** είναι:

- Το *περιβάλλον εκτέλεσης* (Execution Environment Layer),
- Το *επίπεδο αρθρωμάτων* (Module Layer),
- Το *επίπεδο διαχείρισης κύκλου ζωής* (Life Cycle Management Layer), και
- Το *επίπεδο υπηρεισών* (Service Layer)

Υπάρχει επίσης ένα ακόμη επίπεδο το οποίο είναι συνυφασμένο με όλα τα παραπάνω και αφορά στην ασφάλεια του framework.

Execution Environment Layer Τα περιβάλλοντα εκτέλεσης στο **OSGi!** framework αποτελούν συμβολικές αναπαραστάσεις των **JRE!**s (**JRE!**s) που χρησιμοποιούνται κατά την μεταγλώττιση και τη λειτουργία μιας εφαρμογής. Έτσι, αντί να χρειάζεται μια συγκεκριμένη έκδοση και ένα συγκεκριμένο περιβάλλον εκτέλεσης, μπορεί κανείς να θεωρήσει ότι μια εφαρμογή χρειάζεται το περιβάλλον που παρέχει μια τυπική εγκατάσταση του **J2SE!** (**J2SE!**) 1.4 ή του **J2SE!** 6.0. Το σύστημα μπορεί να χρησιμοποιήσει μια υπάρχουσα συγκεκριμένη εγκατάσταση ενός **JRE!** για να παρέχει υποστήριξη για το συγκεκριμένο περιβάλλον εκτέλεσης. Εξάλλου, οι καλογραμμένες εφαρμογές - bundles έχουν αμελητέες εξαρτήσεις από το περιβάλλον εκτέλεσης, οπότε η χρήση τους είναι εφικτή σε μια μεγαλύτερη γκάμα συνθηκών.

[2] `http://www.eclipse.org/legal/epl-v10.html`

[3] Διαθέσιμο στην τοποθεσία `http://download.eclipse.org/eclipse/equinox/`.

Module Layer Η βασική μονάδα διάρθρωσης του λογισμικού που ορίζεται στο **OSGi!** framework είναι το bundle. Ένα bundle ουσιαστικά ενθυλακώνει όχι μόνο μεταγλωττισμένες κλάσεις Java, αλλά πιθανόν και επιπρόσθετα **JAR!** από τα οποία εξαρτάται η λειτουργία του. Ο Java class loader είναι ο κύριος μηχανισμός για την φόρτωση κώδικά από ένα bundle, και σε αυτή τη διαδικασία ακολουθεί σαφείς κανόνες για να φορτώσει ένα υποσύνολο των κλάσεων και των άλλων resources που είναι διαθέσιμα στο bundle. Το Module layer προσφέρει διασύνδεση ανάμεσα σε κλάσεις με ένα ντετερμινιστικό και αυστηρά προκαθορισμένο τρόπο. Κάθε bundle μπορεί να εξάγει (export) και να εισάγει (import) ένα σύνολο κλάσεων στο χώρο ονομάτων του, και το framework επιλέγει από ποιο bundle θα φορτώσει κάποιο πακέτο αν υπάρχει αλληλοεπικάλυψη, ενώ παράλληλα σέβεται εξαρτήσεις σε άλλα πακέτα που είναι δηλωμένες στο manifest file του bundle.

Life Cycle Management Layer Το επίπεδο διαχείρισης του κύκλου ζωής των bundles προσφέρει ένα **API!** για τη διαχείριση των αρθρωμάτων κατά τη διάρκεια εκτέλεσης του framework. Συγκεκριμένα το **API!** αυτό παρέχει τις παρακάτω λειτουργίες:

- *Εγκατάσταση* (install) ενός bundle, που προετοιμάζει το bundle για την εκτέλεσή του παράλληλα με τα άλλα components που είναι ήδη εγκατεστημένα στο framework.
- *Έναρξη* και *Παύση* ενός bundle. Κατά την έναρξη (Start) ορισμένα από τα resources του bundle καθίστανται διαθέσιμα, ενώ κατά την παύση (Stop) γίνεται εκκαθάριση των στοιχείων που δεν χρειάζονται πλέον. Σε μια **OSGi!** Service Platform, όλες οι εφαρμογές μοιράζονται το ίδιο στιγμιότυπο του **JVM!** (**JVM!**), εξοικονομώντας μνήμη και επεξεργαστική ισχύ.
- *Ενημέρωση* (Update) ενός bundle. Το **OSGi!** framework σταματά την εκτέλεση ενός υπάρχοντος bundle και το αντικαθιστά με την ενημερωμένη έκδοση. Έπειτα το ξεκινά και είναι έτοιμο προς χρήση, χωρίς να χρειαστεί να επανεκκινηθεί το **JVM!**.
- *Απεγκατάσταση* (Uninstall) ενός bundle. Όταν ένα bundle δεν είναι πλέον απαραίτητο, ο κώδικας και τα άλλα resources του απομακρύνονται από το σύστημα χωρίς να επηρεάζεται η λειτουργία των άλλων.
- *Παρακολούθηση* (Monitoring) των bundles. Το **API!** παρέχει εκτεταμένες δυνατότητες για την παρακολούθηση της κατάστασης του framework και των bundles.

Το επίπεδο διαχείρισης του κύκλου ζωής βασίζεται σε ορισμένες βασικές οντότητες. Όταν ένα bundle ξεκινά ή σταματά, ο έλεγχος περνά από το framework σε μια οντότητα που ονομάζεται *bundle activator*. Ο activator δεν είναι παρά μια προγραμματιστική διεπαφή που υλοποιείται από μια κλάση του bundle και επιτρέπει τον έλεγχο της διαδικασίας εκκίνησης και παύσης της λειτουργίας του. Μια άλλη σημαντική οντότητα είναι το *bundle event*, που αναπαριστά συμβάντα τα οποία προκύπτουν κατά τις λειτουργίες του ελέγχου ζωής του bundle. Αντίστοιχα, ο *bundle listener* είναι μια οντότητα η οποία ειδοποιείται από το framework όταν προκύπτουν bundle events. Υπάρχουν επίσης αντίστοιχες οντότητες που αφορούν στο συνολικό framework, και ονομάζονται *framework event* και *framework listener*. Τέλος υπάρχει και το *system bundle*, ένα εικονικό bundle που αναπαριστά όλο το framework.

Service Layer Τα bundles μπορούν να συνεργαστούν μεταξύ τους μέσω παραδοσιακών τρόπων όπως η κοινή χρήση κλάσεων. Τέτοιες τεχνικές ωστόσο δε μπορούν να λειτουργήσουν καλά σε ένα περιβάλλον όπου τα bundles (και κατ'επέκταση ο κώδικας που περιέχουν) μπορέι δυναμικά να εγκατασταθούν, να φορτωθούν και να διαγραφούν από το framework. Το service layer ορίζει ένα δυναμικό μοντέλο συνεργασίας που είναι άρρηκτα συνδεδεμένο με το life cycle management layer και παρέχει τη δυνατότητα εύκολου διαμοιρασμού αντικοιμένων ανάμεσα σε bundles. Οι υπηρεσίες δεν είναι παρά αντικείμενα Java που μπορούν να εγγραφούν σε περισσότερες από μια διεπαφές μέσω ενός μητρώου υπηρεσιών.

Προκειμένου να μπορέσουν τα bundles να δημοσιεύσουν, να βρουν και να χρησιμοποιήσουν τις υπηρεσίες που προσφέρει το κσθένα, παρέχεται ο *service mechanism*. Πρόκειται για ένα μηχανισμό που επιτρέπει αυτές τις λειτουργίες και μπορεί να χειριστεί αλλαγές στο εξωτερικό περιβάλλον, αλλά και να υποστηρίξει αλλαγές στις εκδόσεις των bundles, χάρη στη χρήστη *persistent identifiers*, μόνιμων αναγνωριστικών που επιτρέπουν στα bundles να εντοπίσουν κάποια υπηρεσία ακόμα και μετά από την επανεκκίνηση του framework.

Η βασική οντότητα που στηρίζει τη λειτουργία αυτού του επιπέδου είναι η υπηρεσία (*service*). Ένα service είναι ένα αντικείμενο που καταχωρείται στο μητρώο υπηρεσιών του framework (*service registry*), σε μία ή περισσότερες διεπαφές, συνοδευόμενο από κάποιες ιδιότητες. Έτσι κάθε υπηρεσία καθίσταται προσβάσιμη από κάθε bundle που έχει φορτωθεί στο framework. Υπάρχουν επίσης οντότητες αντίστοιχες με αυτές που συναντώνται στα άλλα επίπεδα, όπως τα *service event* και *service listener*.

Security Layer Το security layer βασίζεται στην αρχιτεκτονική ασφάλειας της Java 2. Αυτή παρέχει την έννοια των αδειών (*permissions*) που προστατεύουν πόρους από συγκεκριμένες ενέργειες. Κάθε bundle έχει ένα σύνολο από permissions, και για να επιτρέψει πρόσβαση σε ένα πόρο, μια κλάση ζητά από τον Java Security Manager να ελέγξει αν υπάρχει η απαραίτητη άδεια, προστατεύοντας έτσι τον πόρο από χρήση από κακόβουλους χρήστες.

Επιπλέον, το **OSGi!** παρέχει τη δυνατότητα να μαρκαριστούν κάποια πακέτα που περιέχονται σε ένα bundle ως ιδιωτικά (private), καθιστώντας τα αόρατα και απροσπέλαστα από άλλα bundles. Παράλληλα παρέχει μέσω του μητρώου υπηρεσιών του service layer και της οντότητας *service permission* τη δυνατότητα να εξασφαλιστεί ότι μόνο επιλεγμένα bundles μπορούν να προσφέρουν ή να χρησιμοποιούν συγκεκριμένες υπηρεσίες.

5.2 Axis2 Framework

Το Axis2[4] είναι μια υλοποίηση ανοικτού κώδικα του **W3C!** (**W3C!**) **SOAP!** (**SOAP!**) submission. Αναπτύσσεται υπό την εποπτεία του Apache Software Foundation (ASF) [5] και παρέχει μια υλοποίηση σε Java και C++ του **SOAP!** server, μαζί με βοηθητικές κλάσεις για τη δημουργία **SOAP!** clients. Προέρχεται από τον πλήρη επανασχεσιασμό του ευρέως χρησιμοποιούμενου Apache Apache EXtensible Interaction System (Axis) **SOAP!** stack. Μπορεί να χρησιμοποιηθεί ως αυτόνομη εφαρμογή server και παρέχει τις παρακάτω δυνατότητες:

[4] Το Axis2 είναι διαθέσιμο στη διεύθυνση `http://ws.apache.org/axis2/`

[5] `http://www.apache.org/`

- Αποστολή, λήψη και επεξεργασία μηνυμάτων **SOAP!**
- Αποστολή και λήψη μηνυμάτων **SOAP!** με attachments
- Δημιουργία web service από μια απλή κλάση Java
- Ανάκτηση της περιγραφής **WSDL!** (**WSDL!**) για μια υπηρεσία
- Αυτόματη δημιουργία κλάσεων για server και client από μια περιγραφή **WSDL!**

Το Axis2 είναι σταθερό και τα **API!**s του αλλάζουν σχετικά αργά. Επιπλέον παρέχει το δικό του απλό object model που ονομάζεται AXIOM. Ο πυρήνας του είναι ανεξάρτητος από τις τεχνολογίες μεταφοράς που χρησιμοποιούνται, οπότε μπορεί να χρησιμοποιηθεί για το σχεδιασμό επικοινωνίας μέσω **SOAP!** senders και listeners πάνω από διάφορα πρωτόκολλα, όπως **SMTP!** (**SMTP!**) και **HTTP!** (**HTTP!**).

Για τη βελτιστοποίηση της απόδοσής του σε σχέση με το Axis, το Axis2 χρησιμοποιεί το **StAX!** (**StAX!**) για την ανάλυση των μηνυμάτων **SOAP!**. Παράλληλα επιτρέπει μεγάλη ευελιξία στον προγραμματιστή αφού επιτρέπει τη χρήση επεκτάσεων (extensions) στη μηχανή και τον ορισμό επαναχρησιμοποιήσιμων τμημάτων κώδικα για το χειρισμό μηνυμάτων και την υλοποίηση επεξεργαστικών προτύπων. Τέλος, παρέχει ένα Eclipse plug-in που διευκολύνει στη δημιουργία περιγραφών **WSDL!** από κλάσεις Java και το αντίστροφο.

5.3 Apache Tomcat

Ο Apache Tomcat[6] είναι μια εφαμρογή ανοιχτού κώδικα που μπορεί να φιλοξενήσει και να εκτελέσει Java servlets και να επεξεργαστεί ιστοσελίδες που περιλαμβάνουν κώδικα **JSP!** (**JSP!**). Αποτελεί την πρότυπη υλοποίηση για τις προδιαγραφές των Java Servlet και **JSP!**, και μπορεί να χρησιμοποιηθεί είτε αυτόνομα χάρη στον ενσωματωμένο web server του ή σε συνεργασία με άλλους web servers. Επιπλέον είναι cross-platform εφαρμογή και μπορεί να εκτελεστεί σε κάθε σύστημα για το οποίο υπάρχει διαθέσιμο ένα **JRE!**.

6 Συμπεράσματα

Ένας από τους βασικούς στόχους που καθορίζουν το σχεδιασμό της πλατφόρμας TRACER είναι και η επεκτασιμότητα του, μέσω λειτουργικών αρθρωμάτων. Υπάρχει μια πληθώρα εργαλείων ανάλυσης κώδικα για τον εντοπισμό τρωτοτήτων που χρησιμοποιούνται το τελευταίο διάστημα. Για την ενσωμάτωση τους στην πλατφόρμα, ορίσαμε μια προγραμματιστική διεπαφή (**API!**) καθώς και την κατάλληλη υποδομή, προκειμένου εξωτερικά εργαλεία να μπορούν να ενσωματωθούν στην πλατφόρμα, αρκετά εύκολα. Ο ορισμός των λειτουργιών έγινε σύμφωνα με τις απαιτήσεις που είχαν τεθεί στην φάση της ανάλυσης απαιτήσεων.

[6] Διαθέσιμος στην τοποθεσία `http://tomcat.apache.org/`

Για την προγραμματιστική διεπαφή με τις εφαρμογές πελάτη, αποφασίστηκε η χρήση διεπαφών web service (web service interfaces) ενώ η επικοινωνία μεταξύ του προγράμματος πελάτη και της πλατφόρμας θα γίνεται μέσω του πρωτοκόλλου HTTP και θα ακολουθεί το πρότυπο REST. Αυτή η σχεδίαση μας δίνει την δυνατότητα, να κρυφτούν αρκετές λεπτομέρειες του **API!** που θα παρέχει η πλατφόρμα, κάνοντας την ανάπτυξη των διεπαφών της πλατφόρμας αλλά και εφαρμογών περισσότερο εύκολή.

Ακολουθώντας τις λειτουργικές απαιτήσεις που είχαν τεθεί, ορίστηκαν οι λειτουργίες της προγραμματιστικής διεπαφή που θα χρησμοποιούν τα plugins για την επικοινωνια τους με την πλατφόρμα καθώς και ο ορισμός των επιμέρους κλάσεων.

Αναφορές

[1] Add-ons for Firefox. `http://addons.mozilla.org/en-US/firefox`.

[2] Moodle plugins directory. `http://moodle.org/plugins/`.

[3] NetBeans Plugin Portal, NetBeans IDE Plugins Repository. `http://plugins.netbeans.org/`.

[4] New and Updated Solutions | Eclipse Plugins, Bundles and Products - Eclipse Marketplace. `http://marketplace.eclipse.org`.

[5] Nathaniel Ayewah and William Pugh. The Google FindBugs fixit. In *Proceedings of the 19th international symposium on Software testing and analysis*, ISSTA '10, pages 241--252, New York, NY, USA, 2010. ACM.

[6] Nathaniel Ayewah, William Pugh, J. David Morgenthaler, John Penix, and YuQian Zhou. Evaluating static analysis defect warnings on production software. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE '07, pages 1--8, New York, NY, USA, 2007. ACM.

[7] Brian Cole, Daniel Hakim, David Hovemeyer, Reuven Lazarus, William Pugh, and Kristin Stephens. Improving your software using static analysis to find bugs. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 673--674, New York, NY, USA, 2006. ACM.

[8] Markus Dahm. Byte Code Engineering. In *IN JAVA-INFORMATIONS-TAGE*, pages 267--277. Springer-Verlag, 1999.

[9] David Hovemeyer and William Pugh. Finding bugs is easy. *SIGPLAN Not.*, 39(12):92--106, December 2004.

[10] Haihao Shen, Sai Zhang, Jianjun Zhao, Jianhong Fang, and Shiyuan Yao. XFindBugs: eXtended FindBugs for AspectJ. In *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE '08, pages 70--76, New York, NY, USA, 2008. ACM.

[11] Diomidis Spinellis and Panagiotis Louridas. A framework for the static verification of api calls. *J. Syst. Softw.*, 80(7):1156--1168, July 2007.

Ακρωνύμια

IDL Interface Definition Language

NDFA Non-Deterministic Finite Automaton

ASF Apache Software Foundation

Axis Apache EXtensible Interaction System

Axis2 Apache EXtensible Interaction System 2

EPL Eclipse Public License

Λειτ. απαίτηση	Περιγραφή απαίτησης	Λειτουργία	URL
1.1	Δημιουργία Χρηστών	Δημιουργία Χρήστη	`tracer.usermanagement?act=adduser&username=<username>&password=<password>&type=<type>&name=<name>&surname=<surname>&email=<email>`
2.1	Προσθήκη/Αφαίρεση έργων στη λίστα παρακολούθησης		`?act=addProjectToObservedProjectList&listId=<id>&jarProjectPath=<path>&srcProjectPath=<path>&proj_name=<name> ?act=removeProjectFromObservedProjectList&proj_index=<project_index>`
3.1	Έλεγχος του κώδικα για αδυναμίες που μπορεί να οδηγήσουν σε ρήγματα ασφαλείας	Εκτέλεση της ανάλυσης	`?act=detectVulnerabilitiesOnObservedProjectList&listId=<id>`
3.2	Καταγραφή προβλημάτων ασφαλείας	Καταγραφή	`?act=setLibraryApplication&treatVulnerability=<0|1>`
5.1	Θωράκιση μέσω βιβλιοθηκών ασφαλείας		`?act=addSecurityLibraryToProject&secLibrary=<lib>`

Πίνακας 1: Συσχέτιση απαιτήσεων και λειτουργιών της πλατφόρμας με URLs

Υποσύστημα - Άρθρωμα	Τεχνολογίες
Plug-ins	OSGi
Υπηρεσίες	OSGi, Axis2,
Εφαρμογές Πελάτη	Tomcat, Axis2, JSP, HTML, OSGi

Πίνακας 2: Τεχνολογίες που θα χρησιμοποιηθούν για τα δομικά στοιχεία της πλατφόρμας.