JAVA PATHFINDER (JPF)

Villányi Balázs

Bevezetés

- Java bytecode verifikálására használható keretrendszer
- 1999: A Robust Software Engineering Group a Nasa Ames Research Center
- 2005: open source
- Explicit állapotmodell ellenőrző
  - Egy speciális JVM, amely végrehajtja az adott programot minden lehetséges módon
- http://babelfish.arc.nasa.gov/trac/jpf

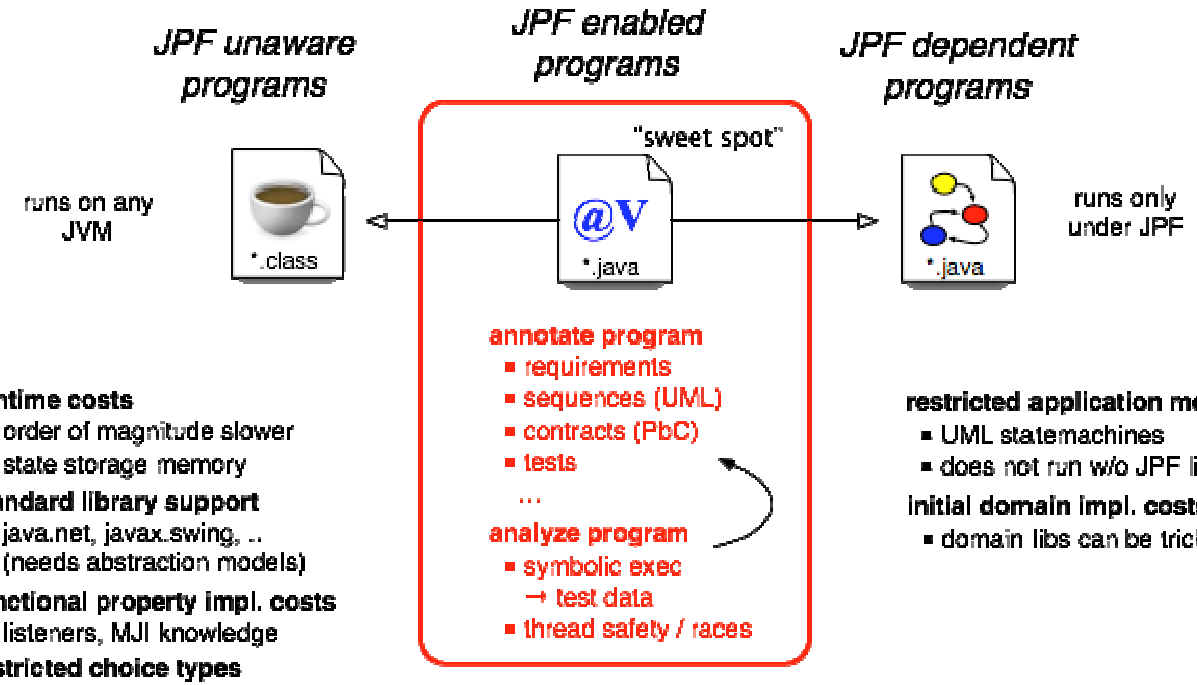
Főbb képességek

- Explicit állapottér modell ellenőrzés
  - Helyi változók, objektomok állapotkövetése
- Szimbolikus végrehajtás
  - Változó értékektől függő végrehajtási alternatívák
- Állapot illesztés
  - Felesleges ismételt végrehajtás elkerülésére
- Visszakövetés (Előző állapotok visszaállítása)
  - több végrehajtási útvonal érdekében
- Részleges rendezési redukció
  - A szálak közötti szükségtelen váltás kiszűrésére
- Hatékony állapottér mentés
  - Saját állapot reprezentáció
- JPF bájtkód végrehajtás
  - Gyors állapot feltérképezés, lassú szekvencia végrehajtás

Mire használható?

- Általános jelenségek vizsgálat
  - Nulla referencia, deadlock, assertion violation
- Állapottér bejárás és vizsgálat
  - Az állapottér robbanás megelőzése
    - Részleges rendezési redukció
    - Szimmetrikus redukciók
      - Osztály betöltés, heap, szálak
    - Heurisztikus állapottér bejárások
    - Szimbolikus végrehajtás
- Kibővíthetőségre tervezett architektúra és konfigurálhatóság
  - Saját funkciók implementálása

JPF alkalmazások



constraints

runtime costs
- order of magnitude slower
- state storage memory

standard library support
- java.net, javax.swing, ..
  (needs abstraction models)

functional property impl. costs
- listeners, MJI knowledge

restricted choice types
- scheduling sequences
- java.util.Random

restricted application models
- UML statemachines
- does not run w/o JPF libraries

initial domain impl. costs
- domain libs can be tricky

benefits

non-functional properties
- unhandled exceptions
  (incl. AssertionError)
- deadlocks
- races

improved inspection
- coverage statistics
- exact object counts
- execution costs

low modeling costs
- statemachine w/o layout hassle,..

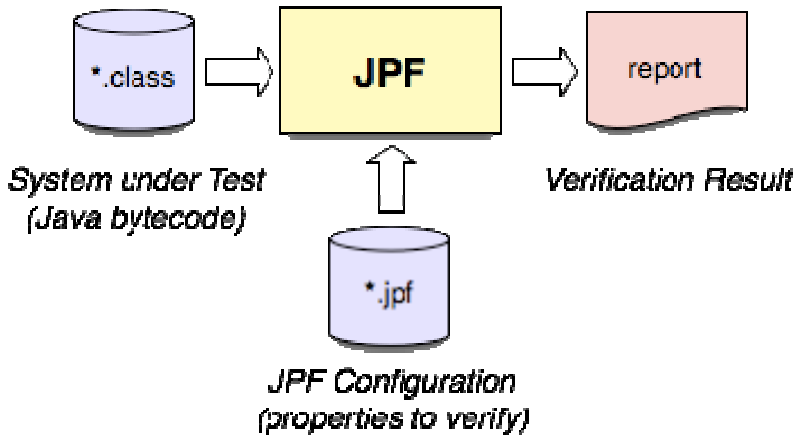
functional (domain) properties
- built-in into JPF libraries

flexible state space
- domain specific choices
  (e.g. UML "enabling events")

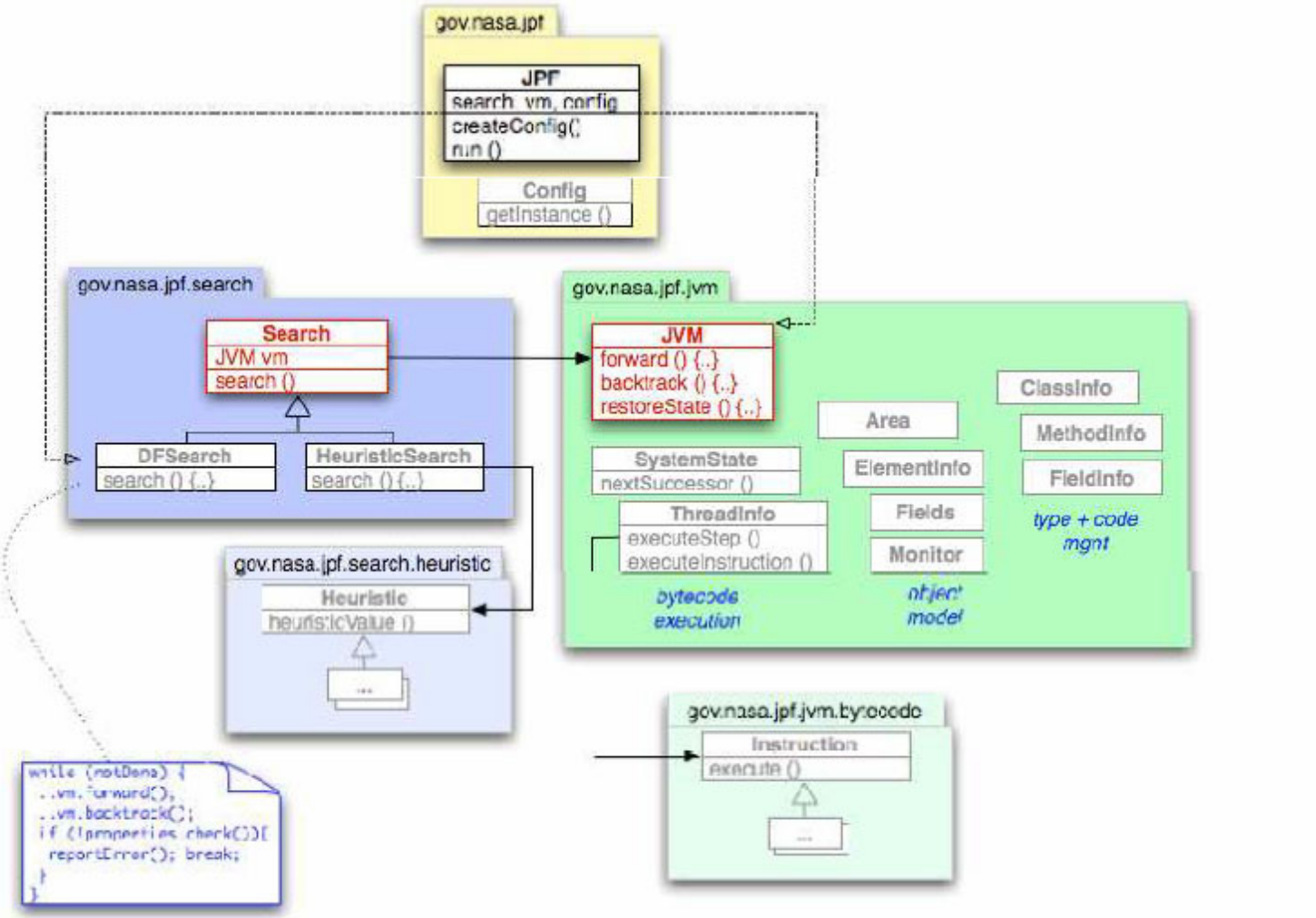
runtime costs & library support
- usually not a problem, domain
  libs can control state space

Végrehajtási modell

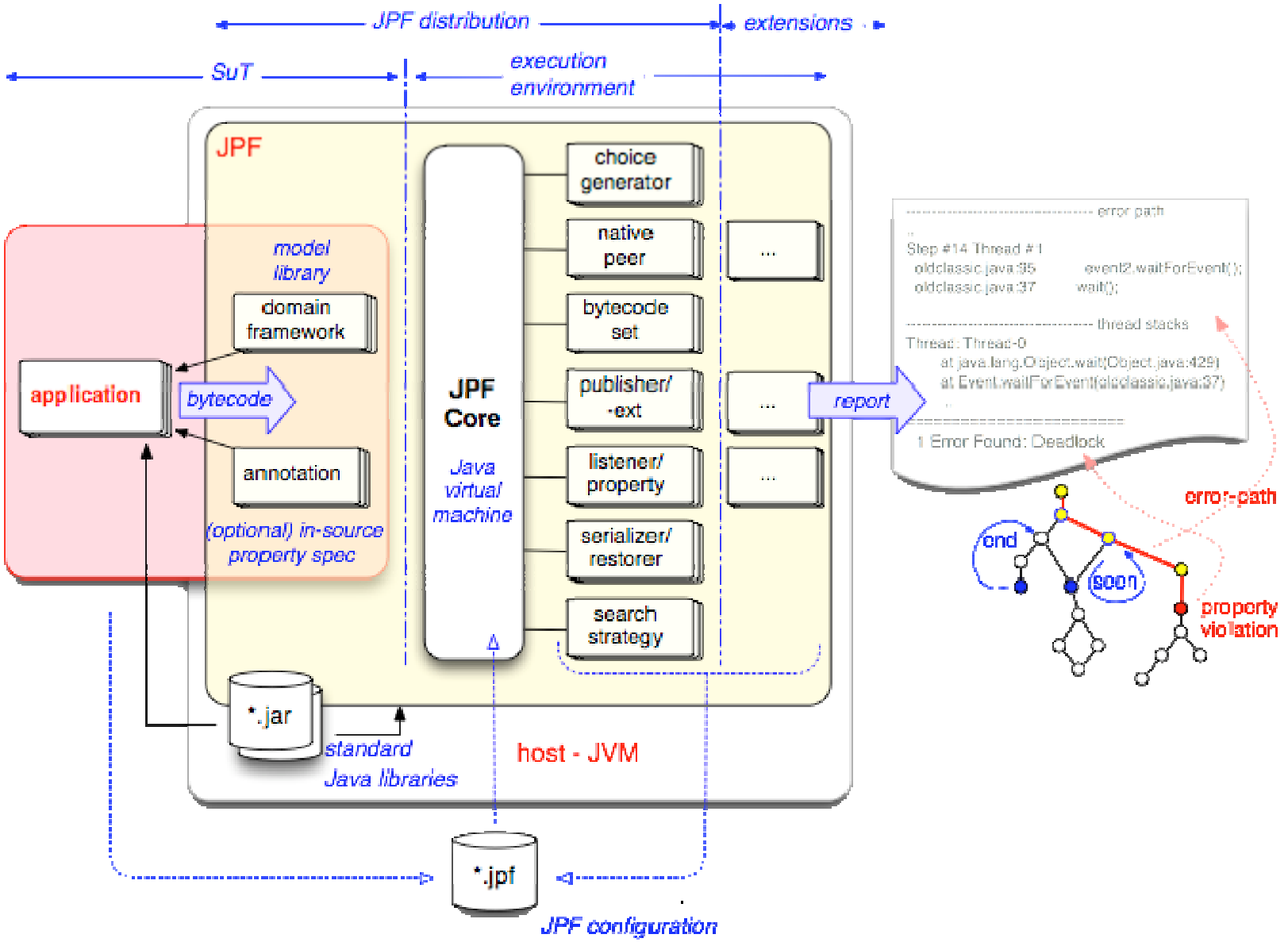
- Végrehajtásban elágazási opciók vizsgálata
- *.jpf konfiguráció
  - Nem-funkcionális tulajdonságok
    - Deadlock és kezeletlen kivételek
  - Saját listenerek
    - Pl. objektumok létrehozása, versenyhelyzet, stb.



JPF állapottér bejárásának implementációja



JPF komponensek

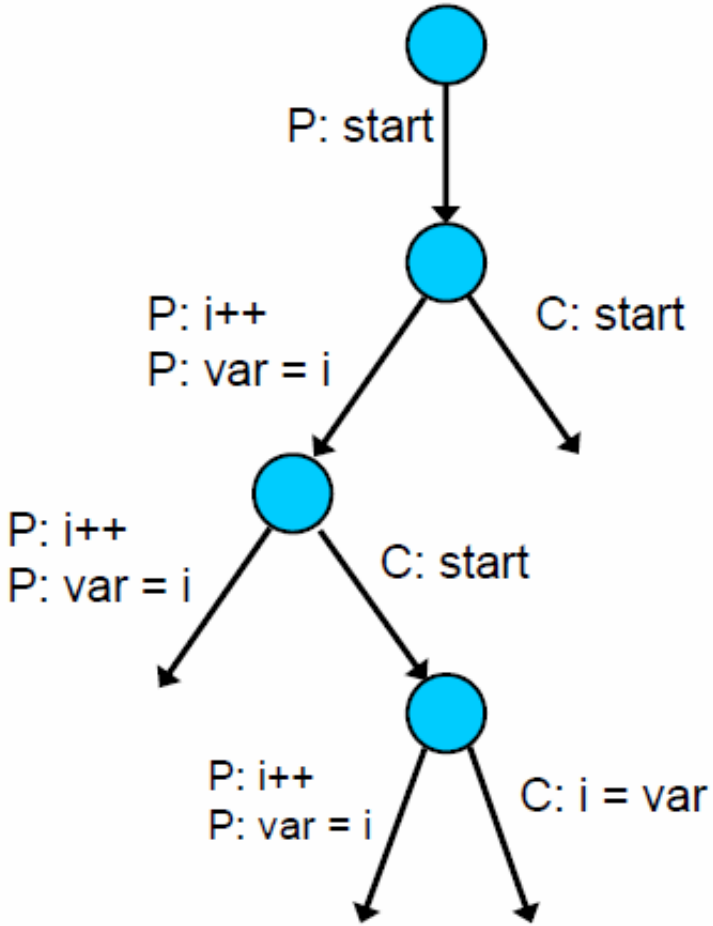


A programállapot tárolása

- Az aktuális állapottér
  - A program JPF reprezentációja
- Az aktuális végrehajtási útvonal
  - A teljes program állapotának reprezentációja egy integer tömbben
  - Inkrementális diffekben tárolja az állapotváltozást
- Hash-mapet készít a már bejárt állapotokból
  - Az állapot összehasonlítások megkönnyítésére

Állapottér generálás menetközben


```
public Producer extends Thread {
  void run() {
    while (true) {
      Main.var = ++i;
    }
  }
}

public Consumer extends Thread {
  void run() {
    while (true) {
      i = Main.var;
      System.out.println(i);
    }
  }
}

public class Main {
  public static int var;

  public static void main(...) {
    (new Producer(var)).start();
    (new Consumer(var)).start();
  }
}
```

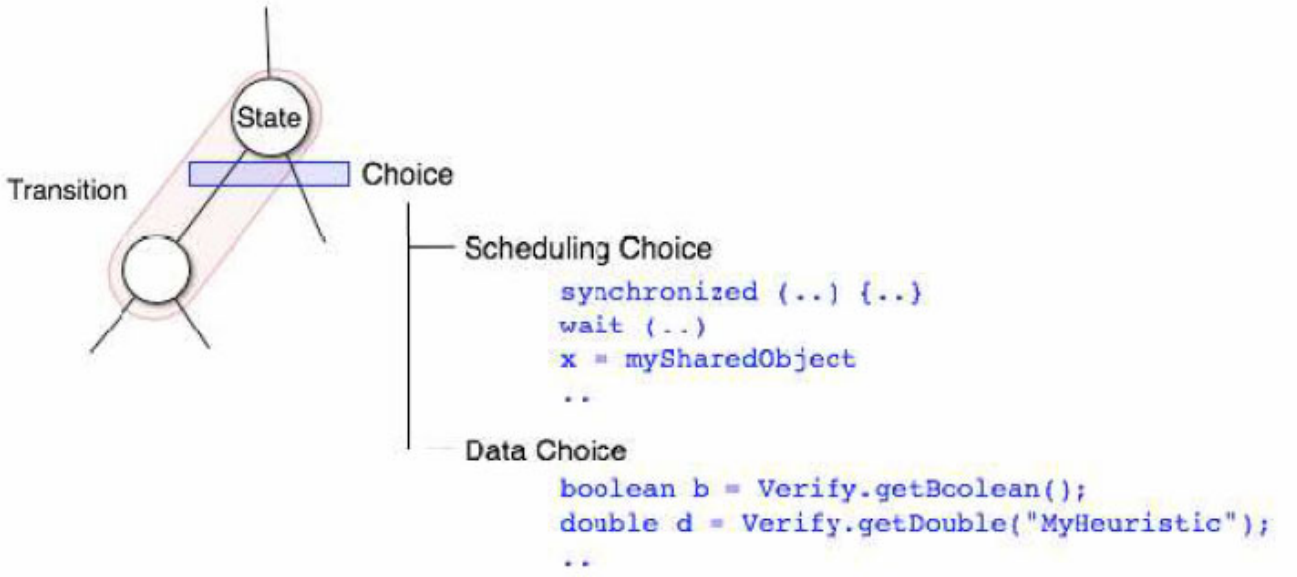


Gyakorlati részleges rendezési redukció

- A modell verifikáció szempontjából szükségtelen szálak közötti váltások elkerülésére
  - A bájtkód utasítások szálon belüli hatásának vizsgálata
- Csak az ütemezés szempontjából fontos bájtkód parancsok vizsgálata
- Előny: gyors megoldás
- Probléma: nem előretekintő
  - A gyémánt alakú bejárási útvonalakat nem tudja kezelni

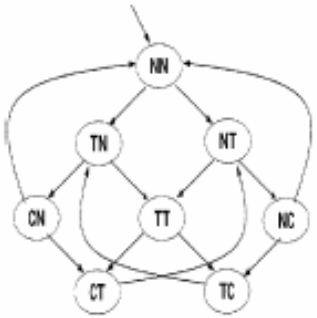
Choice Generator (CG)

- Végrehajtási alternatívák bejárására
  - Közös absztrakció definiálása
- Eredeti motiváció: a nem determinisztikus adatválasztás és szálütemezés egységesítése
- A CG objektum kezeli a (nem) bejárt útvonalakat
  - Az elágazási feltételekkel együtt



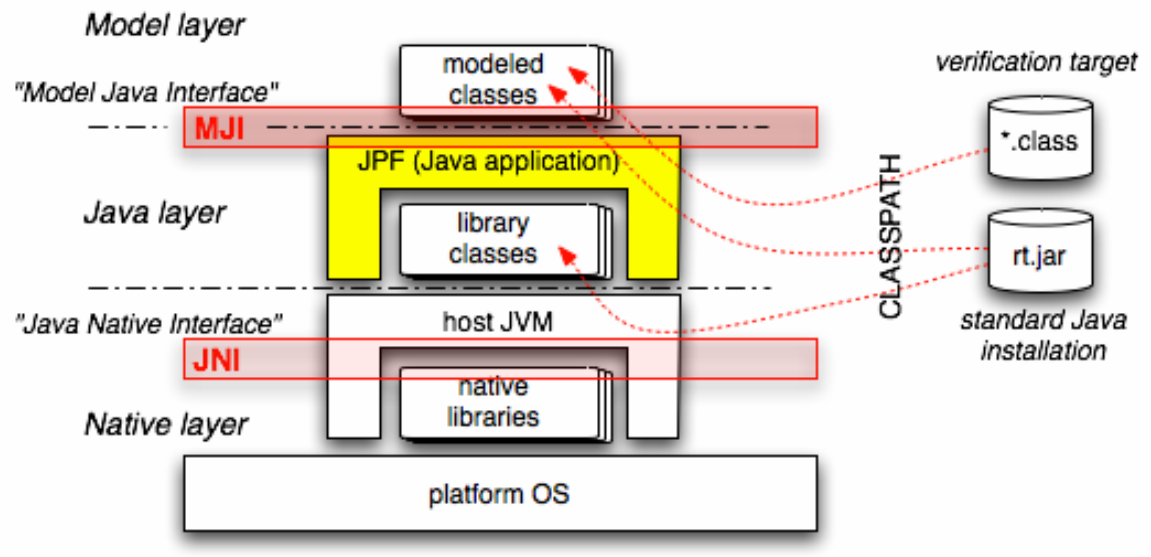
Szimmetria redukció

- Osztály betöltési szimmetriák
  - Az első betöltéskor használt osztálybetöltési sorrend alkalmazási mindenhol
- Heap szimmetriák
  - Az objektumok kanonikus címzését használja
    - Az első betöltéskor kerül meghatározásra forráskód szinten egy külön szálban



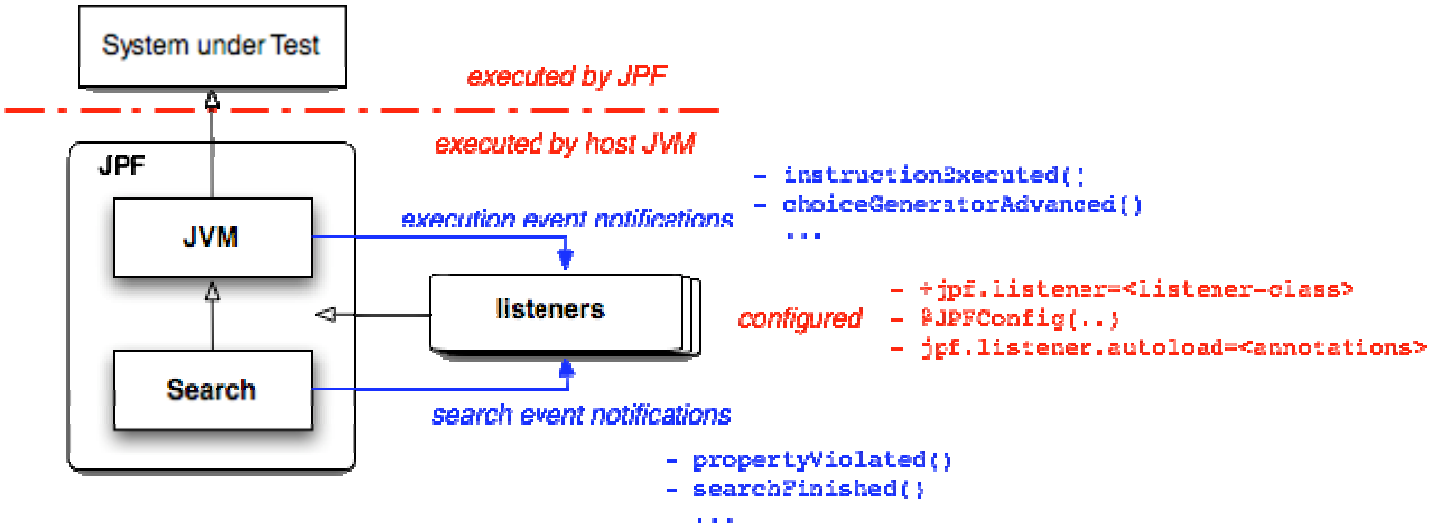
Model-Java Interface (MJI)

- Lehetővé teszi a natív metódusok végrehajtását a (külső) JVM-ben
  - Például: fájl I/O, GUI, hálózat
    - Nincs JPF implementációjuk
- Hasonlít a JNI-hoz



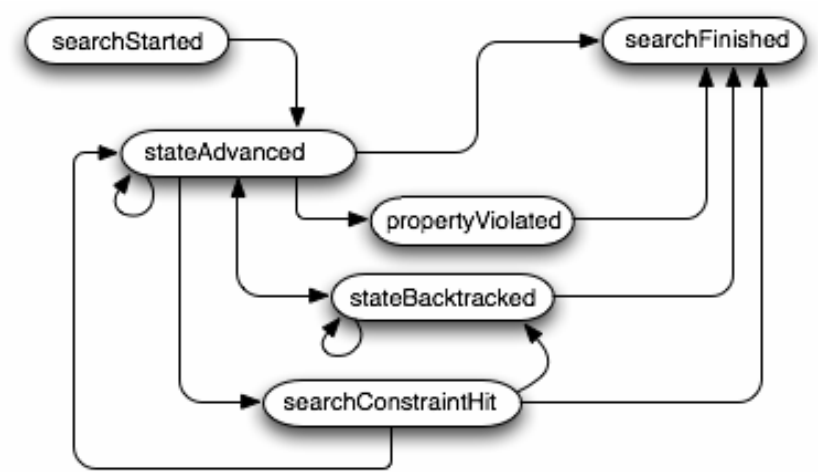
Listenerek

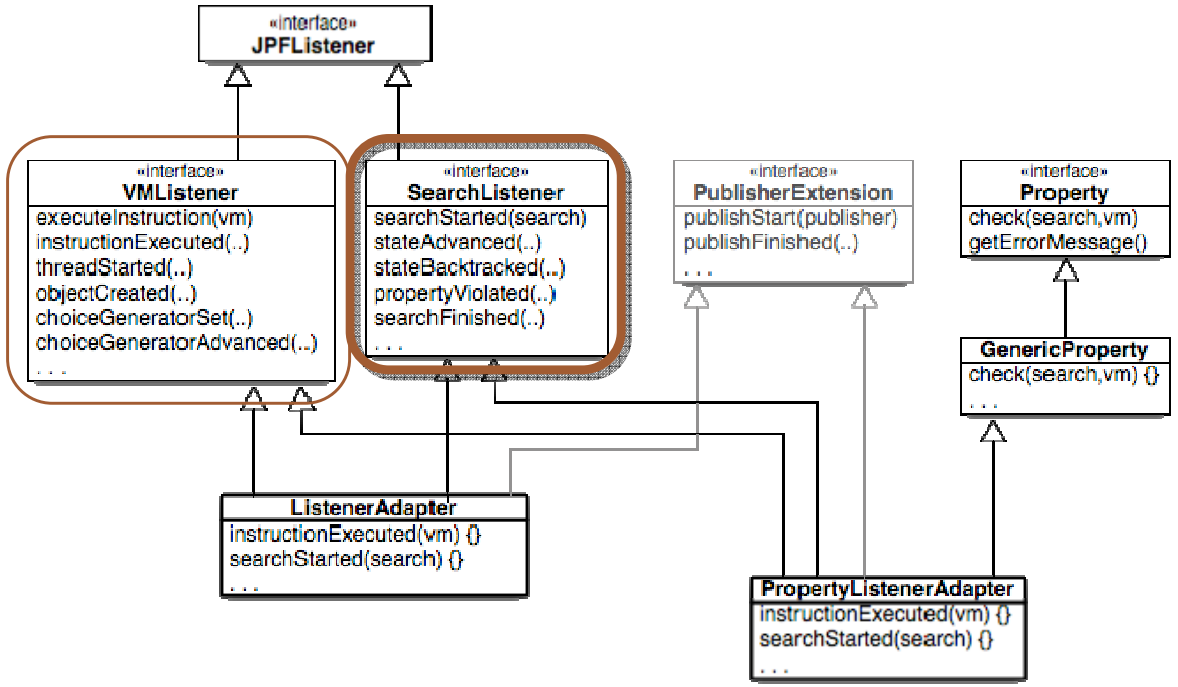
- Legfontosabb kiterjesztési pont
  - A JPF végrehajtás befolyásolása
- Megfigyelő architektúrális minta
- `Listener` property-n keresztül konfigurálható (CL-ből vagy *.jpf fájlból)
- *@JPFConfig* annotáció is használható



Listener típusok - SearchListener

- Állapottér feltérképezés során





VMListener

- Paramétere a JVM objektum
  - Teljes hozzáférést biztosít az éppen vizsgált Java program állapotához
    - Változók, heap, hívási lánc, szálak, GC, CG
- Használata
  - Bájtkód végrehajtás monitorozása
  - A vizsgált kód állapotellenőrzése
  - A hívási lánc vizsgálata
  - Változók nyomon követése

A Verify osztály

- Nem determinisztikus változó meghatározásra

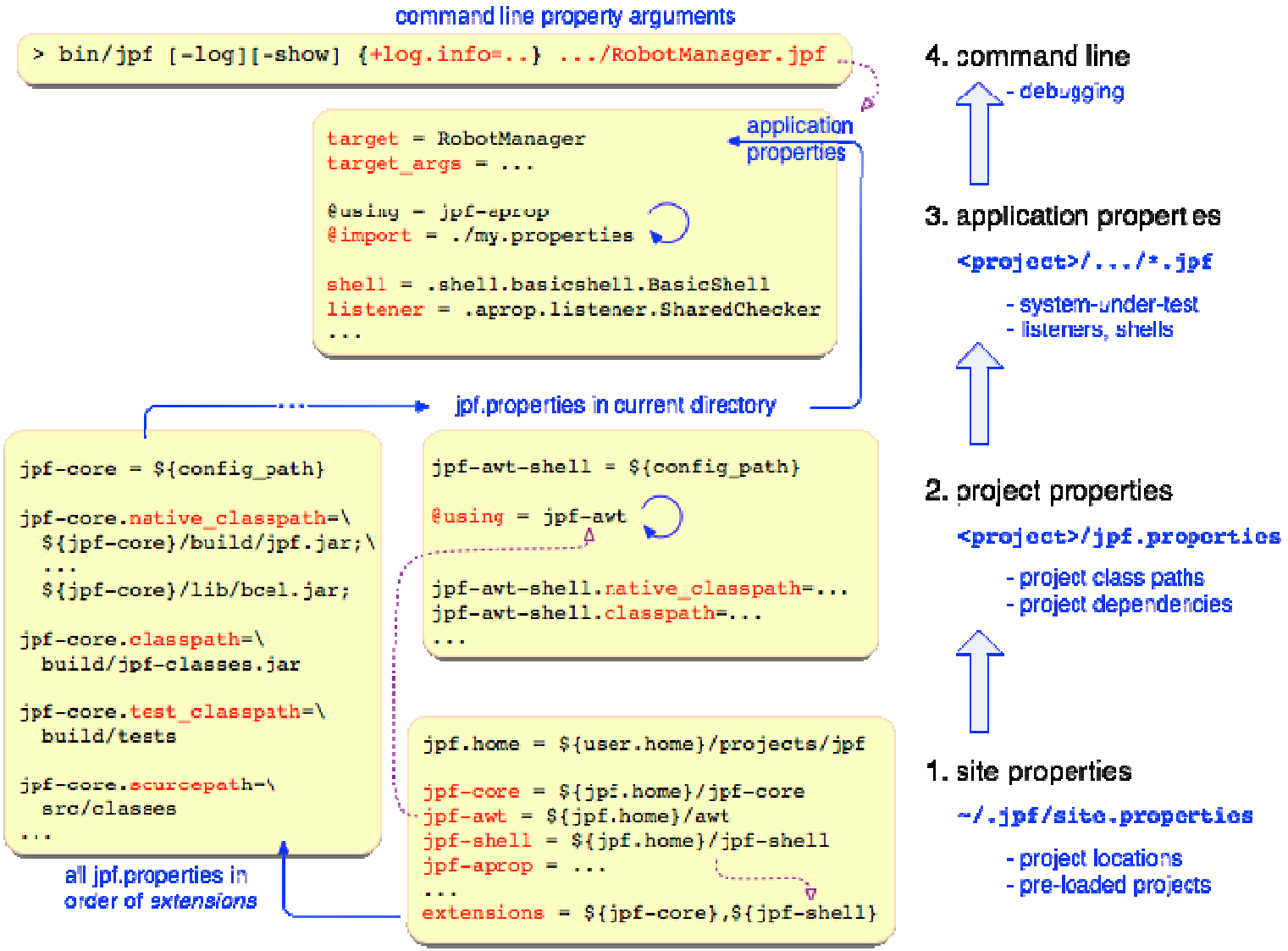
```
import gov.nasa.jpf.jvm.Verify;
...
boolean b = Verify.getBoolean();
if (b) { ... }
else { ... }
...

int x = Verify.getInt(0, 10);
```

  - Támogatott adattípusok: boolean, int, double, Object
- A keresési tér csökkentésére

```
...
Verify.ignoreIf(cond);
...
```

- JPF dependent tesztkörnyezet kialakítására

JPF konfiguráció



Legfontosabb továbbfejlesztések

- Szimbolikus végrehajtás (JPF-SE)
- GUI keretrendszer (JPF-Shell)
- RTSJ és SCJ programok (RTEmbed)
  - Valósidejű és biztonságkritikus alkalmazások
- Hálózati alkalmazások (Net-IOCache)
- UML állapotdiagram modellellenőrzés
- Eclipse, Netbeans IDE támogatás

Összes kiterjeszthetőségi lehetőség.

- Listeners
  - komplex vizsgálatok implementálására
- Peer classes
  - A kód végrehajtása a host JVM-n (JPF VM-n kívül)
- Bytecode factories
  - Alternatív végrehajtási szemantika adott bájtkódhoz
- Choice generators
  - Állapottér elágazások megvalósítása
- Serializers
  - Program állapot absztrakciók megvalósítása
- Publishers
  - Különböző kimeneti formátumok megvalósítása
- Search policies
  - Különböző állapottér bejáró algoritmusok

Köszönöm a figyelmet! – Q&A

Felhasznált források

- Java PathFinder main page @ http://babelfish.arc.nasa.gov/trac/jpf/wiki/intro/classification
- Pavel Parizek - Java Pathfinder General Overview and Current State of Affairs @ http://plg.uwaterloo.ca/~olhotak/seminars/PParizek-JPF.pdf
- Pavel Parizek et al. - Java PathFinder @ http://d3s.mff.cuni.cz/teaching/program_analysis_verification/files/seminar02.pdf
- Symmetry Reduction Methods for Model Checking. Alice Miller and Alastair Donaldson.