STATIC DETECTION OF COMMON BUGS IN JBOSS APPLICATION SERVER

Pavel Vyvial
Master Degree Programme(2), FIT BUT
E-mail: xvyvia00@stud.fit.vutbr.cz

Supervised by: Zdeněk Letko
E-mail: iletko@fit.vutbr.cz

ABSTRACT

This paper describes static detection of common bugs in JBoss Application Server. First, a few bugs from a list of common bugs were chosen and patterns describing these bugs were inferred. Then, detectors searching for such patterns were implemented as plug-ins for FindBugs static analyzer and applied to a development version of JBoss Application Server.

1. ÚVOD

Společnost *Red Hat* se snaží minimalizovat množství chyb ve svých produktech. Z tohoto důvodu vznikla práce přinášející *detektory vybraných častých chyb JBoss aplikačního serveru* patřícího do portfolia firmy *Red Hat.* Detekce chyb je přitom postavena na základech *statické analýzy* podporované nástrojem *FindBugs*.

2. VYUŽÍVANÉ TECHNOLOGIE

JBoss aplikační server [1] je open source implementace Java EE služeb, za pomoci využití na služby orientované architektury. V podstatě se jedná o sestavu předkonfigurovaných *JBoss Enterprise Middleware* komponent (poskytující např. zasílání zpráv, transakce, webový server, bezpečnost, atd.). Celý projekt využívá *bug tracking systém JIRA* pro správu chyb. Pro některé z nich byly v praktické části práce vytvořeny detektory.

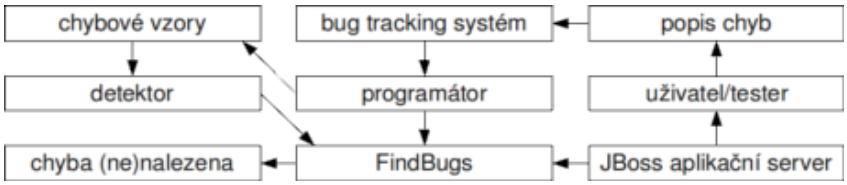
Statická analýza [2] je založená na získávání znalostí ze zdrojového kódu aplikace (popř. byte-kódu) bez jeho spouštění. Na základě zdrojového kódu se odvozuje chování dané aplikace, ve kterém se hledají potenciální chyby. Mezi výhody statické analýzy je možné zařadit: rychlost, schopnost analyzovat obrovské systémy ale i jejich malé části bez potřeby modelování okolí a využitelnost už v brzkých fázích vývoje. Její nevýhodou je časté generování velkého počtu hlášení o chybách, které ve skutečnosti neexistují (tzv. f*alse alarms*). Statická analýza zahrnuje mimo jiné následující druhy: hledání chybových vzorů, analýza toku dat, abstraktní interpretace. V práci bylo z těchto druhů využito *hledání chybových vzorů*. Tento druh analýzy je postaven na myšlence, že podobný druh chyb je

možné popsat *vzorem* (sekvencí po sobě jdoucích příkazů nebo událostí). Analýza založená na *hledání chybových vzorů* umožňuje absolutně garantovat, že určitá analyzovaná aplikace neobsahuje chyby popsané vzorem. To lze považovat za výhodu oproti obyčejnému testování, které tuto skutečnost u některých druhů chyb garantovat nedokáže.

FindBugs [3] představuje platformně nezávislý nástroj využívající statickou analýzu (postavenou na BCEL reprezentaci analyzovaného systému), k hledání chyb v Java programech. Nástroj pracuje nad Java byte-kódem. Je napsaný v programovacím jazyce Java a mimo 300 předdefinovaných detektorů umožňuje i tvorbu vlastních. Detektory se píší v programovacím jazyce Java a do nástroje jsou importovány formou pluginu. Této možnosti bylo využito v praktické části práce.

3. STATICKÁ DETEKCE ČASTÝCH CHYB JBOSS APLIKAČNÍHO SERVERU

Obrázek 1 představuje celý proces od zjištění chyby, přes vytvoření detektoru až po jeho použití. Na tomto obrázku vidíme uživatele, který objeví chybu v JBoss aplikačním serveru. Vytvoří popis chyby a vloží ho do bug tracking systému. Programátor následně tento záznam vyhledá a vytvoří pro něj chybový vzor. Tento vzor pak implementuje jako plugin (detektor) nástroje FindBugs. Když je detektor chyby vytvořen, může ho předložit statickému analyzátoru společně se systémem, ve kterém chce chyby nalézt a získá odpověď zda v systému další podobné chyby existují či ne.



Obrázek 1: Zjištění chyby → popis chyby → chybový vzor → detektor → detekce chyb

Pro práci byl z bug tracking systému získán seznam častých chyb, ze kterého bylo následně k implementaci detektorů vybráno 5 chyb popsatelných *chybovými vzory*. V současné době jsou hotovy dva z pěti detektorů, které budou níže popsány. Všechny zvolené chyby představují porušení interních pravidel při využívání tříd JBoss aplikačního serveru. Každý detektor má jako vstup *byte-kód analyzovaných tříd*. Výstupy detektorů představují *chybová hlášení* o případných detekovaných chybách. Všechny detektory při práci využívají průchod *grafem toku dat* (CFG). Některé z detektorů vytvářejí i *graf volání* (CG).

CFG je orientovaný graf reprezentující všechny možné cesty analyzovaným programem. Uzly *CFG* jsou *basic blocky* (nedělitelné sekvence byte-kódových instrukcí) a hrany jsou určeny pomocí toho jak při spouštění programu na sebe jednotlivé *basic blocky* navazují.

CG je orientovaný graf reprezentující vztahy mezi metodami. Uzly *CG* představují jednotlivé metody a hrany jsou určeny podle toho jak se metody mezi sebou volají. V *CG* tedy vystupují zdrojové a cílové metody.

První detektor funguje následujícím způsobem: 1) Pro všechny analyzované třídy se provede kontrola, zda třída patří do projektu ESB. Pokud se třída v ESB nenachází, je třída přeskakována. 2) Prochází se přes CFG všech metod od tříd patřících do ESB projektu a hledá se `invoke` instrukce. 3) Při nalezení `invoke` instrukce se testuje, zda je cílem nebez-

pečná metoda. Pokud ano, nahlásí se chyba společně s místem, kde k nalezení nebezpečného volání došlo. Jinak se pokračuje dokud se neprohledají všechny instrukce metody.

Druhý detektor funguje následujícím způsobem: 1) Průchodem přes CFG se vytvoří CG nad všemi analyzovanými třídami. Po vytvoření CG obsahuje každý jeho uzel jedno z následujících označení: `safe`, `unknown`, `hasCourierInterface` a `inaccesible`. Označení `inaccesible` obsahují uzly nedosažitelných metod (analýze nebyl poskytnut jejich byte-kód). Označní `hasCourierInterface` získají uzly metod patřících do tříd implementujících jednu z tříd `DeliverOnlyCourier` nebo `PickupOnlyCourier`. Pokud nějaký zdrojový uzel hrany v CG nepatří do třídy `Invoker`, tak označení `hasCourierInterface` získají i cílové uzly hran grafu patřící do třídy `CourierFactory`. Označení `safe` bývá přiděleno uzlům korespondujícím s metodami patřícími do třídy `ServiceInvoker.` Označení `safe` dále získají i uzly metod odkud není přes `invoke` instrukci volán nějaký uzel s označením `hasCourierInterface`. Všechny ostatní uzly nevyhovující ani jednomu výše popsanému označení získají označení `unknown`. 2) Cílové uzly CG s označením `hasCourierInterface` propagují své označení zdrojovým uzlům. Propaguje se tak dlouho, dokud je kam (vytvoří se tranzitivní uzávěr). 3) Všechny uzly, které získaly označení `hasCourierInterface` propagací se přeznačí na `danger`. Tyto uzly představují množinu metod obsahující potenciální chyby.

4. ZÁVĚR

Po nastudování problematiky bylo vytipováno pět častých chyb. Pro první dvě z těchto chyb byl implementován detektor jako plugin statického nástroje FindBugs. Testy, které byly s detektory prováděny zatím nenašly nový výskyt těchto chyb. První detektor ovšem našel dva výskyty chyby, kterou popisuje ve starší verzi třídy EJBProcessor komponenty Rosetta verze 4.4 projektu JBossESB. Práce bude pokračovat implementací zbývajících detektorů a jejich případným zobecnění pro možnou znovupoužitelnost u jiných chyb.

PODĚKOVÁNÍ

Tato práce vznikla částečně za podpory grantu VUT FIT, FIT-S-10-1 a specifického výzkumu MSM0021630528. Na závěr bych chtěl ještě poděkovat svému vedoucímu Ing. Zdeňku Letkovi a zaměstnanci společnosti Red Hat Ing. Jiřímu Pechancovi za jejich připomínky a čas, který mi věnovali.

REFERENCES/LITERATURA

[1] JBoss community. Jboss Application Server Installation And Getting Start Guide. [Online], [rev. 2009-11-17], [cit. 2010-02-28]. Dostupné na URL: https://www.jboss.org/community/docs/DOC-12923

[2] F. Nielson, H.R. Nielson, and C. Hankin. Principles of Program Analysis, Springer-Verlag, 2005.

[3] Bill Pugh, David Hovemeyer, and Ben Langmead. FindBugs – find bugs in java programs. [Online], Verze 1.3.9 (2009), [rev. 2009-08-22], [cit. 2010-03-17]. Dostupné na URL: http://findbugs.sourceforge.net/